Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Esa Tanskanen

Transition synthesis for skeletal animations using optimization and simulated physics

Master's Thesis
Espoo, June 14, 2014

Supervisor: Professor Perttu Hämäläinen
Advisor: Professor Perttu Hämäläinen

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Esa Tanskanen

Title:
Transition synthesis for skeletal animations using optimization and simulated physics

Date:	June 14, 2014	**Pages:**	73
Major:	Media Technology	**Code:**	IL3011
Supervisor:	Professor Perttu Hämäläinen		
Advisor:	Professor Perttu Hämäläinen		

Despite the large advances in digital animation and 3D modelling techniques, producing digital 3D animation is still a labour-intensive task and therefore expensive, even when using motion capture technologies. These high costs can be especially prohibitive for small gaming studios, many of which have emerged as smartphones have became common. High-quality animations are sold in several online marketplaces, such as Mixamo, but transitions from one animation to another still need to be created separately. Commonly used transition methods require that the source and destination animations of the transition can be blended together procedurally, which may not be the case with animations sold in a shop. Therefore the results may look highly unrealistic.

Automatic motion generation and transition synthesis have been studied for decades. Even so, only the recent methods have not been either limited to specific kinds of animations, such as walking, or limited to highly simplified physical models which are unsuitable for modelling humans. The present methods still do not produce as natural looking motion as motion capture or hand-made animation. They also require tuning of several control parameters, which may not be intuitive for the user. After the control parameters have been changed, the generation process has to be run again, which may take a long time, from minutes up to several hours. Many of the methods are also not easily parallelized for performance gain. Besides, it is common that they may not find the most optimal solution.

In this thesis, generating transition animations between arbitrary source and destination animations using spacetime optimization methods has been studied. In order to speed up the motion generation and to allow the user to quickly try different control parameter values, a sequential importance sampling based optimization method is used for solving the spacetime optimization problem. To further increase the performance, the system is also highly parallelized to utilize the multiple cores in modern computers. Finally the benefit of using a sequential sampler over a non-sequential one for faster control parameter tuning is evaluated.

Keywords: animation, motion synthesis, motion planning, physical simulation, optimization, stochastic optimization

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Esa Tanskanen

Työn nimi:
Transitioanimaatioiden synteesi fysiikkasimulaation avulla

Päiväys:	14. Kesäkuuta 2014	**Sivumäärä:**	73
Pääaine:	Mediatekniikka	**Koodi:**	IL3011
Valvoja:	Professori Perttu Hämäläinen		
Ohjaaja:	Professori Perttu Hämäläinen		

Vaikka 3D-animaatio- ja mallinnustekniikat ovat kehittyneet huomattavasti, 3D-animaatioiden luominen vaatii edelleen huomattavasti ihmistyötä ja on siksi kallista, vaikka käytettäisiin liikkeenkaappaustekniikkaa. Kustannukset voivat olla ennen kaikkea kohtuuttomat pienille pelialan yrityksille, joita on syntynyt paljon älypuhelinten yleistymisen myötä. Korkealaatuisia animaatioita myydään useissa internet-kaupoissa, kuten Mixamossa, mutta siirtymät animaatiosta toiseen täytyy silti luoda erikseen. Yleisesti käytetyt siirtymäanimaatiomenetelmät vaativat, että lähde- ja kohdeanimaatiot voidaan sulauttaa yhteen ohjelmallisesti, mikä ei ole välttämättä mahdollista kaupasta ostettujen animaatioiden kohdalla. Lopputuloksena voi olla erittäin epäluonnollisen näköinen animaatio.

Automaattista liikkeen ja siirtymäanimaatioiden tuottamista on tutkittu vuosikymmenten ajan. Silti vasta viimeaikoina on julkaistu menetelmiä, jotka eivät ole rajoittuneet tietyn tyyppisiin animaatioihin, kuten kävelyyn, tai jotka eivät käytä erittäin yksinkertaistettuja fysiikkamalleja jotka eivät sovellu ihmisten mallintamiseen. Nykyiset metodit eivät edelleenkään tuota yhtä luonnollista animaatioita kuin liikkeenkaappaus tai käsin animoiminen. Ne vaativat myös useiden ja mahdollisesti käyttäjälle epäintuitiivisten kontrolliparametrien säätämistä. Kontrolliparametrien muuttamisen jälkeen liikkeentuottamisprosessi täytyy ajaa uudelleen, mikä voi kestää kauan, muutamista minuuteista jopa tunteihin. Monet menetelmistä ovat myös vaikeita rinnakkaistaa, mikä heikentää niiden tehokkuutta. Sen lisäksi ne eivät välttämättä löydä optimaalisinta ratkaisua.

Tässä diplomityössä on tutkittu siirtymäanimaatioiden automaattista tuottamista vapaasti valittavien lähde- ja kohdeanimaatioiden välillä spacetime optimization -menetelmän avulla. Liikkeen tuottamisen nopeuttamiseksi ja jotta käyttäjä voisi nopeasti kokeilla eri kontrolliparametreja, sekventaaliseen importance sampling -menetelmään perustuvaa optimointimenetelmää on käytetty spacetime optimization -ongelman ratkaisemiseksi. Järjestelmän tehokkuutta on pyritty kasvattamaan rinnakkaistamalla järjestelmän toimintaa huomattavasti, jotta saataisiin suurin hyöty nykyisten prosessorien useista ytimistä. Lopuksi on arvioitu sekventaalisen samplerin hyötyjä kontrolliparametrien hienosäätämisessä.

Asiasanat:	animaatio, liikesynteesi, liikkeen suunnittelu, fysiikkasimulaatio, optimisaatio, stokastinen optimisaatio
Kieli:	Englanti

Acknowledgements

Thanks for the project team, Perttu Hämäläinen, Sebastian Eriksson and Sami Hakkarainen, for introducing me to the academic way of working. And thanks for Emilia Hänninen for going through the trouble of proofreading a publification written by an engineer.

Espoo, June 14, 2014

Esa Tanskanen

Contents

Chapter 1

Introduction

Digital 3D animation is widely used by film and gaming industries to produce believable animated characters, but it remains highly labor intensive. Depending on the quality requirements and the complexity of the animation, Milic and McConville [2006] estimated that an animator can generally be expected to produce only 3 to 7 seconds of digital animation in a week. An alternative method is to transform the motion of a live actor to a digital animation with motion capture technologies. However, skilled actors and expensive equipment are required, which may cost too much, e.g. for a small gaming studio. Regardless of how the animation is made, those animations are also strongly tied to a specific environment, e.g. depend on the character standing on a flat ground, and Rose et al. [1998] found out that animations are commonly difficult to modify to suit a new environment.

In games, the controllability of the character demands that the motion of the character should adapt to the control signals from the player. A widely used method is to produce animations for each possible state of the character, such as walking and sitting, and also create a transition animation for each possible transition between a pair of states. However, since the required number of possible transitions often increases rapidly when the amount of character states increases, the required number of transition animations becomes huge, significantly increasing the amount of animation work required. Therefore, automated methods for generating transition animations have been researched for decades, for example by Brotman and Netravali [1988], Rose et al. [1998] and Sung et al. [2005].

Hundreds of research papers have been released on automated motion synthesis, and Hodgins and Wooten [1998] estimated already 16 years ago that it would not take long for human motion to be simulated in real-time. However, despite the large advances in the technology, Al Borno et al. [2013] asserted very recently that computer synthesis of natural-looking human mo-

tion is still out of reach. The resulting motion is generally not physically correct, as in the work by Rose et al. [1996], or it lacks stylistic properties, as in e.g. the work by Al Borno et al. [2013]. However, Hodgins and Wooten [1998] stated that short transitions between man-made animations should be much easier for computers to generate than longer animated sequences. This is because transition animations are commonly relatively short and generally not containing important keyframes that define the basis of the motion. If successful, Hodgins and Pollard [1997] estimated that automated generation of natural-looking transition animations could save countless of work hours and allow rapid prototyping of new characters and actions.

1.1 Goals of Thesis

The thesis was part of a project studying the benefits of stochastic optimization methods in motion synthesis. Therefore the goal of the thesis was to use stochastic optimization to build as automatic and generic transition generation system as possible using state-of-the-art methods. The motion generation was to be as automatic as possible, as was the goal of Al Borno et al. [2013]. Rose et al. [1998] found out that such a system should also require minimal tuning and intuitive control parameters. Learning to use it should also be as easy and straightforward as possible, as was also the aim of Liu and Popovic [2002]. If adjustments have to be made to the animation, it should have been possible to tune the animation on the fly, as Sung et al. [2005] found that to be important for animators. The environment of the animation was supposed to also be easily changeable and generating a transition was intended to be possible regardless of what point in time during the playback of the state animations the transition should start or end. The resulting animation, created offline, should be ready to be used by any game using the Unity game engine with any Mecanim-based human character rig. The system was based on an existing framework which had been built as a part of the research project. The framework contained a physics simulator, a stochastic optimizer and an interface to build optimization goals and evaluate trajectories. The following sections explain each of these functionalities.

1.2 Structure of Thesis

Existing character animation methods and their limitations are introduced by Chapter 2. The basics of physical simulation and how to simulate physical human characters are then covered by Chapter 3. Controlling the simu-

lated character is explained in Chapter 4, and the most important method, spacetime optimization, is covered in Chapter 5. Chapter 6 discusses various challenges of synthesizing lifelike or artistic motion. Chapter 7 explains how motion synthesis can be interpreted as an optimization problem, and explains the general challenges of optimization methods and the fitness landscape. The spacetime formulation used in this thesis is then explained in Chapter 8. Chapter 9 describes how the transition generation system was implemented. The experiments to test how well the solution adapts to the changing landscape of the optimization problem are introduced in Chapter 10. Conclusions of the results follow in Chapter 11. Chapter 12 explains how the system could be improved in the future and what was left out of consideration.

Chapter 2

Character Animation

Animations bring digital characters to life. The problem with digital character models is that they are usually highly detailed, containing up to millions of polygons in pre-rendered animations. Therefore the model is usually simplified for animating purposes and the animation of the simplified model is then transformed back into the highly-detailed model, as done by e.g. Kavan and Žára [2005]. Parent [2002] stated that working with the simplified model is also easier for the animator as they can work with higher level controls rather than directly transforming the numerous polygons. The granularity of the simplified model depends on the amount of detail required in the animation. The following sections introduce a commonly used simplified character model, the skeletal model. Common ways to produce animation with the model using keyframe and motion capture techniques and some existing ways to blend animations and the limitations of the methods are also shortly discussed.

2.1 Skeletal Animation

A skeletal model splices the character to a graph-like structure of interconnected parts that can be transformed separately. A common way is to have a vertex in the graph for each physical joint in the character that the graph represents. These vertices are called joints and the edges between them called bones. To animate the model, the joints are moved, often also constraining the length of the bones to be constant. There are multiple ways to transform the animation back to the detailed character. The most popular method in real-time animation playback is linear blend skinning (LBS), where each vertex in the detailed model has a mapping for how much the position of each joint influences the position of the vertex of the character mesh. As

the name suggests, the positional influences are simply summed together to find out the final position of the vertex. Therefore, the method, although computationally efficient and easy to implement, is prone to deformations, as found out by e.g. Kavan and Žára [2005]. Since its introduction, various improvements to the algorithm have been suggested to increase the quality of the results, e.g. learning corrective displacements or rotations for the mesh based on example motion data, as done by Wang et al. [2007].

2.2 Keyframe Animation

Techniques similar to digital keyframe animation were already used in traditional, hand-drawn animations. In the pose to pose traditional animation approach, frames with key character poses are identified and the rest of the frames are interpolated between them, often using arc-like motion paths for the body parts of the characters to simulate inertia. Similarly in digital keyframe animation, the animator transforms the skeletal model to the key poses and defines the times when the animation should pass the key poses. The in-between animation frames are calculated with interpolation, using similar methods as in traditional animation such as ease in and ease out which simulate acceleration and slowing down, respectively. The simplicity and efficiency of calculating the in-between frames means that the method is commonly used in both pre-rendered and real-time animation playback. [Parent, 2002]

Pullen and Bregler [2002] found the strength of the traditional keyframe animation techniques to be the artistic freedom that they provide. With a sufficient number of keyframes and fine-tuned interpolation control all kinds of skeletal animation can be presented. However, Liu and Popovic [2002] asserted that the issue with this kind of a freedom is that the physical aspects of the motion are not automatically considered, but must be handled by the animators themselves. This has led to earlier suggestions, such as by Isaacs and Cohen [1987], for a system which would automatically enforce physical correctness of the animation. Regardless of the method how the animation is produced, it may be presented as a keyframe animation as long as enough keyframes are defined.

2.3 Motion Capture

One way to produce physically plausible and realistic animation is by recording it. A commonly used way is to attach either positional sensors or visual

markers to live actors. The positions of the markers are then captured at specific time intervals. In the case of the positional sensors, the surrounding magnetic field is measured to detect the position of the sensor. The locations of the visual markers on the other hand are detected with a large number of cameras. The results have historically been prone to errors. [Parent, 2002]

Moeslund et al. [2006] found out that the accuracy of the methods has improved over time, but there are still issues left with those methods. Even the cheaper of the options, video capture of optical markers, still requires industrial-quality equipment such as multiple cameras, markers and software. While animator workload is reduced, manual work is still required in acting the motion and cleaning up the results. The capturing system also requires repeated calibration. [Parent, 2002]

Overall, Rose et al. [1998] found out that motion capture yields fairly competitive results. Its downside, as stated by Panne and Lamouret [1995], is that it is less suitable for animating anything else than human motion, such as plants, animals or imaginary life forms. The captured motion is also valid only in the environment it was captured in, which is a common issue noted by e.g. da Silva et al. [2008]. Motion which was captured on a flat floor may not work well when played back on a slope or staircase. Modifying the motion afterwards to suit a new environment is also highly difficult, although automated methods for making the modifications have been proposed. For example, Pullen and Bregler [2002] combined motion capture with keyframe animation, in an attempt to provide the artist a way to describe a rough sketch of the animation using keyframe techniques, which the system would then augment using motion capture data as a reference. However, their method does not enforce any physical constraints or even ground contacts.

2.4 Animation Blending

Animation blending is often used in real-time animation, either to mix together multiple existing animations to generate a motion which has the combined characteristics of the mixed motions, or to blend between a source and a target state animation to automatically generate a transition animation. Each source motion is assigned a weight, which tells how much that motion should affect the resulting mixed motion. A transition animation is generated with motion blending by gradually changing the weights of the motions. Assuming that the weights sum to one, first the starting motion has a weight of one and the ending motion a weight of zero. Along the transition the weight of the starting motion is gradually changed to zero while the weight of the ending motion changes to one. Usually this blending is done during

specific, predefined parts of the source and the destination animations. A naïve linear blending method would blend the joint rotations of the skeletal animations together using the gradually changing weights. This may however easily cause serious artifacts in the resulting motion. One example would be blending between walking and running. Since the timing of the footsteps will be different, the results will likely have wiggly leg motions. [Kovar and Gleicher, 2003]

To overcome the timing issues, the source animations may be slowed down or speeded up to match the timings of each other. This method, called time-warping, assumes that the source motions still look fine even with a changed speed, which may not always be the case. Also, in order to automatically match the source and the destination motions, they must be similar enough so that corresponding parts of the motions can be detected. This is not true for most kinds of motions, for example when blending from a running to a crouching motion. Also since parts of the start and end motions are spent in the transition, such a transition method may not work with short motions or when the entire source motions must play from start to end, and the generated motion would take in place between them. [Kovar and Gleicher, 2003]

Even if the source motions are physically correct, Menardais et al. [2004] stated that the issue with motion blending is that the blended result may not be physically correct anymore. Most blending methods, like those by Kovar and Gleicher [2003] and Menardais et al. [2004], only consider some physical aspects like ground contacts as a post-processing step. Therefore physical correctness like conservation of momentum is in most part not taken in account. The resulting motion cannot also contain anything that is not present in the source motions, such as taking extra steps for slowing down, balancing or moving the center-of-mass to a desired point. Overall, the existing blending methods work best when the blended motions are as close to each other as possible. This has led to Rose et al. [1996] to research a physically correct motion blending method, but their method also does not implement physically correct motion of the selected root position of the character (root motion). That method is also mainly suitable for very short transitions taking less than 0.6 seconds.

Chapter 3

Physical Simulation

In a physical simulation the movement of a character is not pre-defined. Instead, the character is typically composed of simulated rigid bodies that have forces and torques acting on them causing the bodies to move. Every object which should have its own path of movement should be modelled by a separate rigid body. Each of the bodies is composed of one of more geometric shapes, called fixtures, such as spheres, cuboids (box shapes), planes, convex polygons, cylinders and capsules, which are cylinders capped with hemispheres. [Isaacs and Cohen, 1987]

In contrast to soft bodies, the shape of the fixtures in rigid bodies cannot change during the simulation, and, as Featherstone [2008] told it, their material seems infinitely hard. These bodies may be connected to each other using virtual joints, in order to build highly complex models such as virtual human models, as done in various works e.g. by Isaacs and Cohen [1987]. If the whole environment is physically modelled with rigid bodies, the motion of the bodies would automatically adapt to the environment. For example, the character could collide with the environmental objects around it and any environmental effects, such as a blowing wind, could affect the character. This way the physical simulation can be used to automatically generate motion sequences. [Liu and Popovic, 2002]

An iterative physical simulation calculates the accelerations of the bodies based on forces, torques and constraints such as joints and collisions. This calculation is called forward dynamics. The simulation then updates the position and rotation of the rigid bodies based on the velocities, accelerations and the length of the simulation time step using numerical integration. Due to being a numerical method, the accuracy of the physical simulation is limited by the rounding errors of the calculations, but also depends a lot on the implementation of the physics engine. [Boeing and Bräunl, 2007]

A widely adopted way is to use sequential solving, which solves one physi-

cal constraint at a time. Because the method does not take in account all the joints and collisions at the same time, solving a subsequential constraint may invalidate any previously solved constraints. Therefore the method has to iteratively repeat the solving phase, usually a pre-defined number of times, and the results may still not be perfect after the iterations as found out by Moravánszky [2005]. To counter this deficiency, linear complementary problem (LCP) based methods, which take in account all the constraints at once, have been introduced to physical simulation. Solving the LCP can be implemented either with iterative methods or pivot-based methods, such as Lemke's Algorithm and its variants, as in the work by Brock et al. [2009]. Pivot-based methods are guaranteed to find a perfect solution unlike iterative methods, but they are generally computationally intensive.

3.1 Joints

In a 3D simulation, free-moving rigid bodies have 3 positional coordinate values and 3 rotational values around the coordinate axes, forming 6 degrees-of-freedom (DOF). The bodies may however be connected to each other by virtual joints. These joints limit the movement of the bodies, effectively reducing their DOF in respect with each other. Some examples of joint types are hinge and ball-and-socket joints. Hinge joints simulate a similar connection as a door has with a wall, allowing the bodies only to rotate relative to each other around a defined axis, local to the system consisting of the two bodies. The distance of the bodies to an anchor point in the local coordinate system is also fixed. Hinge joints therefore allow only a single degree-of-freedom between the connected bodies. When connected with ball-and-socket joints, similar to a human shoulder, the bodies are allowed to freely rotate around the anchor point, forming 3 relative degrees-of-freedom. Also in this case, the distance to the anchor point is fixed. When a set of bodies are connected together by joints, their combined DOF is the summed number of freedoms of its joints plus 6 DOF from the movement and rotation of the set as a whole. [Isaacs and Cohen, 1987]

$$N_{DOF} = d + r + h + 3b \tag{3.1}$$

In formula 3.1, N_{DOF} is the degrees-of-freedom of the object, d is the dimensionality of the space, r is the number of rotational axes, h is the number of hinge joints inside the object and b is the number of its ball-and-socket joints.

3.2 Collision Models

The rigid bodies may also collide with each other, since they have a volume, which is defined by their fixtures. The collisions can be modelled as temporary joint-like links, which disallow the bodies to move into each other but allow them to separate breaking the contact, as in the work by Geijtenbeek and Overmars [2011]. Also when the simulated bodies slide against each other friction forces slow down their movement. Tassa et al. [2012] found out a completely realistic friction model to generally be the most difficult part of a physical simulation, since it would consist of many small-scale events happening in very short spans of time. Therefore, as done by Tassa et al. [2012], the usual way to simulate such phenomena is to collectively evaluate all contact forces that would occur during a longer time scale.

The Coulomb friction model, where the maximum frictional force is directly proportional to the surface normal force acting on the object, was found out by Geijtenbeek and Overmars [2011] to be commonly used. Typically an approximation such as a pyramid-shaped friction cone is adopted due to being linear and therefore simplifying the calculations, as in the work by Mordatch et al. [2012]. An additional simplification, used by Mordatch et al. [2012], was to allow collisions to only occur inside pre-defined areas on the surfaces of the bodies. Furthermore, Al Borno et al. [2013] disabled contacts altogether between certain bodies to further reduce the computing time required by the simulation. However, Liu et al. [2005] assered that the collision model cannot be overly simplified, since an insufficiently detailed contact model may cause unrealistic motion.

3.3 Physical Human Model

Liu et al. [2005] stated that a complex biomechanical system such as a human cannot be perfectly simulated by a computer. However, the most important features of motion can be captured with a skeletal body model similar to the skeletal animation model introduced in Chapter 2.1. The skeletal model, as used by e.g. Isaacs and Cohen [1987], defines the character as rigid bodies for the major body parts. The rigid bodies are connected to each other by simulated hinge, ball-and-socket and universal joints, emulating the actual joints inside the human body. The body parts can be formed with detailed models or with simple geometrical shapes such as capsules. Using simple shapes, as done by e.g. Tassa et al. [2012], is more common since they require less computational resources.

The properties of the body parts, such as length, radius and thickness,

are derived from the details of a real human, as done by Lo et al. [2002]. However, as the model is usually constructed with few geometric primitives, its mass distribution and collision surfaces are not highly accurate. Popović and Witkin [1999] stated that the granularity of the model should depend on how the model is intended to be used. When simulating a robot grabbing an object in its arm, a detailed model of fingers may be needed, but in a higher-level motion, such as jumping, detailed fingers may not be needed to capture the essence of the motion. When modelling in lower detail, multiple parts of the human body are combined into a single, inseparable rigid body. In some cases the model is highly simplified, such as the single chain, horizontally mirrored body model used by Lo et al. [2002]. However, Al Borno et al. [2013] feared that a model which is simplified too much will not have the degrees of freedom required for natural-looking movement.

In a simple ragdoll model none of the joints are actuated, so they are not able to generate forces. However, that can only model a dead or unconscious human. To enable the character model to move itself, all of its joints are usually actuated. Isaacs and Cohen [1987] asserted that realistic form of actuation would require modelling muscles, which provide the forces, tendons, the tissues that join the muscles to the bones, and ligaments, which join the connected bones. Each degree of freedom also requires at least two muscles, because a single muscle can only provide a pulling force. However, such a detailed model is seldom used due to its complexity and computational requirements, as shown by e.g. Stewart and Cremer [1992], Al Borno et al. [2013] and Popović and Witkin [1999].

Instead, joints are commonly modelled with angular servo motors providing direct torques, with a fixed maximum amount of torque they can provide, as in the work by Geijtenbeek and Overmars [2011]. This model does not take in account that, as explained by e.g. McLester and Pierre [2007], muscle forces cause larger or smaller torques depending on the angle of the joint. As illustrated by the Figure 3.1, the rotary component of the muscle force, which causes the torque at the joint, is smaller if the joint angle is larger. In contrast, the torque is always the same when using a servo motor, unless the motor torque would be calculated using a model which would take in account the joint angle. In addition, the angular motor model does not consider the way muscles and tendons store energy when the joint is not in its equilibrium state, due to their spring-like nature, as explained by Kawato [1999]. Liu et al. [2005] also stated that humans also prefer to use certain muscles over others. Therefore, a character modelled with servo motors tends to work in a robot-like manner, and to overcome this Geijtenbeek and Overmars [2011] found out that the focus on muscle-based models has been increasing lately.

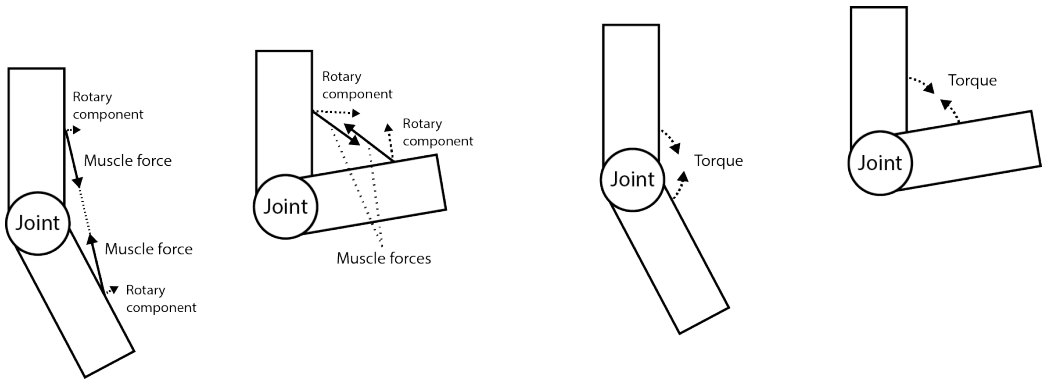

Figure 3.1: Comparison of forces in the simplified muscle model by Isaacs and Cohen [1987] and their rotary components and the torques in the commonly used angular motor model.

Chapter 4

Motion Control

In order for the simulated physical model of a human or animal to move, it requires a controller to decide when to use its motors to provide the desired movement. This controller works like the animal brain, considering the environmental situation and controlling the joints accordingly. The control signal is a function of time, and in case of angular motors, yields the torques acting on the joints at the instant of time. The control of a character is indirect in the sense that the position and the rotation of the character cannot be directly controlled. The movement of the character is the result of the torques acting on its joints and any contact forces caused by being in contact with other bodies such as the ground. In other words, the characters are considered to be underactuated. The following sections introduce the commonly used methods to calculate the torques for the joints required to move the character in the desired way. [Geijtenbeek and Overmars, 2011]

4.1 Joint-Space Control

The control function usually uses its knowledge of the past and present situation and the dynamics of the system. That kind of a control scheme is called a closed-loop, feedback control. One example is the proportional-integral-derivative (PID) controller, which tries to drive a control variable to a desired value based on the present error, the difference of the present state and the desired state, (proportional), the accumulation of the earlier errors (integral) and the predicted change in value (derivative). The controller is often used in motion control, without the integral part, to drive the motors to a desired target pose. As the target pose may change over time, the integral part is not useful as any changes to the target value will invalidate the accumulated error. When each motor has a separate PD controller, it is

a form of joint-space control, where each part of the character is controlled separately and act independently of the other joints. Therefore the amount of coordination between joints is limited. The gains of the PD controllers have to be selected carefully so that the stiffness of the motion realistically matches to the actual stiffness of human muscles for the desired motion. Another method is to use inverse dynamics to calculate the torques required for the bodies to move in a desired way during the next time step, as explained by e.g. Geijtenbeek and Overmars [2011]. Both the PID control and inverse dynamics require however that the desired trajectories of the body parts are already known, so a way to generate those trajectories is also required.

4.2 Procedural Control

The character controller may be designed for a specific situation or kind of movement. Procedural controllers, which work by using a pre-defined set of rules formulated for a specific situation, have been designed for various kinds of motions such as walking, as done by Faloutsos et al. [2001], as well as running, bicycling and vaulting in the work by Hodgins and Wooten [1998]. However, Jain et al. [2009] stated that these controllers generally work only in certain, pre-defined conditions. Transforming the controller to a new environment generally requires either a lot of hand-tuning of control parameters or designing a completely new controller. Building a procedural controller for the desired situation or motion requires a lot of time and specialized knowledge. A method to overcome the limitation, suggested by Faloutsos et al. [2001], was to combine several simpler and more generic controllers into a higher level controller. However, as the goal of this thesis is to build a controller that adapts to all kinds of environments and motions, designing the set of such controllers and ways to combine them remains a difficult task.

4.3 Model-Predictive Control

The dynamics of a physical system are usually highly complicated. If the controller is designed to be purely reactive, only taking in account the present situation and not attempting to make any predictions about the future, Mordatch et al. [2010] stated that the control may appear short-sighted. That happens because a controller trying to maximize the immediate benefit cannot move the character into a worse state, even if the worse state would allow for better movement in the future. An example of this would be jumping: If the target is to make the center-of-mass (COM) of the character to reach as

high point as possible, the character has to crouch first to allow a larger angular motion at the knee joint. However, crouching lowers the COM, initially bringing the character away from the objective. The character controller needs to be able to predict that the crouching is required to allow for the desired kind of jumping motion later on. Therefore, in order to find the correct control signal, the controller has to make anticipatory predictions about the future, in other words, as stated by Geijtenbeek and Overmars [2011], to form a look-ahead policy.

In model-predictive control (MPC) the knowledge of the physical system and its dynamics are used to predict the way bodies will move in the future. Zordan et al. [2007] explained that this allows the character to be alert and anticipate future events. However, predicting the future requires that the planning horizon, the length of the time window that the prediction takes in account, is long enough. The downside is that a longer planning horizon usually means slower computing times. Since we are designing a non-realtime, offline tool, computing times are not as critical as in a real-time case, and the planning window can span the entire duration of the generated transition animation. However this requires that the user has pre-defined the maximum length for the transition animation.

Kawato [1999] argued that using MPC to control animals is supported by biology since the delay before a control signal propagates from the brain to the muscles is very long, the whole control loop taking 150 to 250 milliseconds. Even for spinal-controlled reflexes, the control loop still takes 30 to 50 milliseconds. In comparison, control loops in robotics have very high frequencies, up to 10 000 Hz, and therefore can quickly react to any sudden changes in the situation. With such delays in the animal control loops it is impossible to control the movement solely using corrective actions based on feedback information. Therefore, the brain must have a predictive model to send the appropriate control signal in a feed-forward way, knowing beforehand how the muscles need to be used to move the limbs to the right places. However, aiming with the limbs is also helped in a smaller scale by the spring and dampener characteristics of the tissue and the muscles which also stabilize the motion. Stiffness of the muscles increases stabilization, but is typically reduced as one becomes more proficient in a movement skill. Besides this kind of automatic, spring-dampener based stabilization, another way to do any last moment adjustments to the motion of a simulated character is by coupling MPC control with an adjusting reactive controller as done by Da Silva et al. [2008].

Chapter 5

Spacetime Optimization

While MPC finds out the control parameters required for optimal motion, spacetime optimization, introduced by Witkin and Kass [1988], calculates the optimal trajectories for the positions and the rotations of each part of the character for a defined time window. Therefore the result is not a control signal but instead a fully detailed animation. In contrast to joint-space control, the torques of all the joints are also considered at the same time, allowing for a high level coordination between different parts of the body. The optimization problem was formulated by Witkin and Kass [1988] as a set of goals and constraints. The constraints, such as preventing objects from penetrating each other, must be satisfied for the entire trajectory. From the space of possible solutions that satisfy the constraints, the trajectory that best achieves the goals, such as using minimal motor energy, is selected.

Whether a property of the desired trajectory, e.g. the character staying in balance, should be expressed as a goal or a constraint depends on if the condition it represents must always be perfectly satisfied or not. If the condition must always hold, it should be formulated as a constraint, and if the condition should only be attempted to be satisfied as well as possible it should be expressed as a goal. The main difficulties of the method are selecting the right goals and constraints and searching through the limitless amount of possible trajectories. How well a goal is satisfied is expressed with an objective function or a fitness function. The fitness function returns a value (fitness value), which describes how well the goals are satisfied. The fitness values of the goals need to be combined together to form the overall fitness value of the trajectory, as was done by e.g. Geijtenbeek and Overmars [2011]. During the optimization, the trajectory is expressed with a set of optimized variables. The following sections describe how the constraints, the goals and the trajectory are commonly formed.

5.1 Constraints

Because the motion of the character is modelled with rigid bodies, Geijtenbeek and Overmars [2011] found out that the most common spacetime constraints are the dynamics constraints that arise from the Newton-Euler laws of motion. These constraints ensure that the motion stays physically plausible. In some rare cases as in the works by Liu and Popovic [2002] and Mordatch et al. [2012], the dynamics constraints are modelled as soft constraints, allowing for breaking the laws of physics if a solution cannot otherwise be found. The joints of the character and their limits also form physical constraints. Other physics-based constraints are disallowing rigid bodies from penetrating each other and restricting the friction forces according to the Coulomb friction model, as in the work by Geijtenbeek and Overmars [2011]. Additionally, Sentis and Khatib [2006] enforced a minimum distance between the character and selected physical objects. That way they simulated the natural reaction of keeping the personal space clear around the character. However, to reduce the amount of constraints derived from physical correctness and to speed up calculations, the collisions between different parts of the character's body may not be taken in account, as done in the works by Mordatch et al. [2010] and Wu and Popovic [2010]. Collisions could be then avoided by overly limiting the ranges of the joint angles. However, that would disable motions like leg-crossing, which is important in some balancing tasks, such as those in the work by Wu and Popovic [2010].

If the motion synthesis tries to move the character from a specific starting pose to an ending pose, the poses themselves can also be added as constraints to the optimization, requiring the trajectories to start from the starting pose and end in the end pose. There may also be additional key poses between the start and end, as used by Safonova et al. [2004]. However, since the character is underactuated, it is possible that the end pose cannot be reached exactly, and in this case a solution cannot be found. Therefore it may be better to try to find a trajectory which only ends up as close to the end pose as possible, as explained in the following section. The available time for the character to reach the destination can also be implemented as a constraint, as done by Safonova et al. [2004], or a goal, which was used by Mordatch et al. [2010].

5.2 Goals

The most common goal in spacetime optimization is energy minimization, which has been used since Witkin and Kass [1988] first introduced spacetime optimization. The reason for its wide appeal in spacetime optimization-

based solutions is, as Kawato [1999] explains it, that it approximates the way humans tend to minimize the usage of muscle power. In practice this is commonly done by minimizing motor torques accumulated over the optimized period of time, often by summing squared torques as done by Safonova et al. [2004] and Al Borno et al. [2013]. A similar model is jerk minimization, which minimizes the change of torques, as explained by Van Welbergen et al. [2010]. Other models are using joint accelerations instead of motor torques, a method used by Mordatch et al. [2012], or penalizing velocity of the center-of-mass (COM), as done by Tassa et al. [2012]. Penalizing the velocity of the COM tries to minimize the amount the character will move in general, but does not actually minimize energy consumption, since the limbs can move in several ways without actually moving the COM. Minimizing joint accelerations does not take in account how much mass the movement of a joint will displace and therefore may lead to minimizing the movement of the body parts which do not have a large impact on the general movement of the character. Jerk minimization on the other hand attempts to preserve the stiffness of the muscles during the motion. This has the basis in the observation that the stiffness of the muscles does not vary largely during a motion. [Liu et al., 2005]

Target poses can also be expressed as goals as in Liu and Popovic [2002]. One way to formulate the goal is by minimizing variances between the target and realized positions of the end effectors, hands and feet, as studied by Kawato [1999], which has a basis in the studies of human trajectory planning. Other body parts can also be considered in the goal, and the relative importance of the correct positioning of the body parts can be expressed by having a weight value for each body part in the goal formulation. This weight, as used by Mordatch et al. [2012], would tell how much the position of a body part affects the overall fitness value of the goal. Tracking motion capture animation by minimizing the difference between synthesized motion and the reference animation, as in the work by Muico et al. [2009], could also be viewed as a posing goal. Besides the locations of the body parts, the pose goals may also specify target values for the velocities of the body parts, as in the work by Isaacs and Cohen [1987], which are attempted to be matched in a similar way as the joint positions. Geijtenbeek and Overmars [2011] asserts that using the target velocities has been rare, possibly because building an intuitive interface for specifying them is difficult. Most commonly all joint velocities are defined to be zero at the start and the end of the motion, as done by Lo et al. [2002] and Zordan et al. [2007], and that does not require any specific editing capabilities from the user interface.

Task-dependent goals may also be used, even though they do not easily generalize to other kinds of tasks. For example, Hodgins and Pollard [1997]

had a target for the air time, how long the character is in a state of free fall, Liu and Popovic [2002] formulated target speeds for the feet at specific points of time, Hodgins and Wooten [1998] had a goal for keeping the torso upright and Al Borno et al. [2013] had a target height for the COM. Avoiding damage to vital body parts was also considered in several cases, especially by Zordan et al. [2007]. These task-specific goals may help to specify the best solution to the optimizer, since they are based on observations of what kinds of conditions must hold for the desired motion. Tassa et al. [2012] specifically avoided having motion specific goals in order to keep their optimization method fully automated. In this thesis the aim was also designing a generic controller, so the amount of task dependent goals was to be kept minimal. However, Tassa et al. [2012] regarded solutions not depending on motion specific goals as effectively providing worst-case results and that motion specific goals could be required to further improve the quality of the motion.

The fitness values provided by these goals need to be combined to find out the overall fitness of a candidate trajectory. It is usual that the goals compete with each other, such as how tracking a motion capture clip can interfere with balancing, as in the work by Geijtenbeek and Overmars [2011]. A common way to combine the fitness values is simply by calculating a weighted sum, as done by Al Borno et al. [2013] and Safonova et al. [2004] among others. Often this requires hand tuning of the weights of the summed fitness values, which changes the importance of the goals in relation to each other, as in the methods used by Jain et al. [2009] and Safonova et al. [2004]. Tuning the weights of the parameters and that way changing the outcome of the optimization was viewed by Rose et al. [1998] as giving the user of the system high-level control of the desired motion. However, selecting the correct weights can require a lot of work, even if the values may not have to be exactly tuned to produce a desired motion, as shown by Al Borno et al. [2013]. Therefore Liu et al. [2005] proposed automated ways for finding estimated values from motion capture data. Yin et al. [2008] found out that the difficulty of tuning the parameters is further increased because it can be hard to know how changing the parameters will affect the results and what kind of changes are needed to make the motion look like it should. Hodgins and Pollard [1997] stated that the goals may also be valid for a single character only and need to be tuned or automatically adapted to a new character that has a different size, mass or other fundamental property.

5.3 Trajectory Control Parameters

The optimized variables should be chosen so that they can formulate any kind of a trajectory. When Witkin and Kass [1988] first introduced the concept of spacetime optimization this was not an issue, since they used a simple 2D model with 4 hinge joints for demonstration. The entire object therefore consisted of 6 DOF, 4 from the joints, 2 from the root position and 1 from the root rotation. This is much lower than the commonly used 30-60 DOF human models in 3D space, as in the works by Al Borno et al. [2013] or Safonova et al. [2004]. Therefore if the control parameters describe the states of each joint at every evaluated instance of time, as was done by Witkin and Kass [1988], and the number of time steps and the optimized window of time are large, the number of optimized variables also becomes huge with a high-DOF model.

Another way is to only use the optimized variables to express some key poses along the trajectory and to calculate the control values between these control points with interpolation. Al Borno et al. [2013] formulated a reference trajectory by placing evenly spaced control points inside the optimized time window. Each control point was a represented by a set of joint angles. They calculated the values between the control points by interpolating the key values using cubic B-splines. During the control optimization, which they used instead of spacetime optimization, they used joint-space PD controllers to calculate the torques for the joint motors. They also hand-picked the gain parameters of the PD controllers instead of having them as optimized parameters. However, as explained in Chapter 4.1, the same PD gains may not work for all kinds of motion.

Since the trajectory described by the optimized variables is not followed precisely, the character will likely not actually end up in the reference poses. Instead, the reference trajectory should be viewed as being purely a control signal. This is especially shown in the work by Mordatch et al. [2012], as instead of forming the trajectories for all body parts, they only parameterized the trajectories of the end-effectors at the key frames and calculated the orientation of the rest of the body using inverse kinematics. Liu and Popovic [2002] used a similar method as well, using mass points as reference points instead of end-effectors. The benefit of those methods is the reduced number of control parameters that is required to represent the trajectories, but the disadvantage is that the exact orientation of any other body part than those specified in the control points cannot be expressed.

Auxiliary optimized parameters may also be introduced to aid the optimization process. Mordatch et al. [2012] parameterized the contact locations

between pre-defined body parts and environmental objects. The contact information was treated as being part of the interpolated control points. They stated that the advantage of the method was that contacts are an important part of human motion, and explicitly specifying them rather than treating them as a result of the physical simulation allows the optimizer to manipulate the desired set of contacts directly. The downside is that the optimization finds the set of desired contacts only for the control points, which means that a large number of control points is needed when simulating situations where the set of contacts should change a lot, such as rolling on the floor. Also the areas which can be in contact with other objects have to be specifically defined. Increasing the amount of contact areas or control points also increases the number of optimized parameters, but parts of the body which are not covered with contact areas could penetrate objects.

Chapter 6

Style of Synthesized Motion

When synthesizing human motion the aim is often to make the results to appear as realistic as possible. It has proven to be difficult, as people can easily see if the motion is not perfectly natural, as in the work by Faloutsos et al. [2001]. Hodgins and Wooten [1998] stated that we have such a good eye for even the slightest details of motion that we can identify a person just by looking at the way they walk. This issue is emphasized by the uncanny valley effect, introduced by Mori [1970], which states that nearly human-like motion looks even stranger and repulsive than noticeably robotic or otherwise distinctively non-human-like motion. To measure the realism of a synthesized motion Hodgins and Pollard [1997], Lo et al. [2002] and Liu et al. [2005] compared the results with a video record of a person acting a same kind of a motion. Hodgins and Wooten [1998] called this the Turing test of motion synthesis, meaning that the generated motion is realistic when it is indistinguishable from actual human motion. However, the qualitative evaluations in each of those works have no details about the actual data collection methods, such as how many people participated in the research and how the sessions were conducted. This may suggest that the focus has been in developing novel methods instead of actually evaluating them. It seems that quantitative evaluation, also used by most other works regarding motion synthesis, gives more tangible results, even if it has been difficult to express realism with quantitative measurements. Therefore the evaluation of the results of this thesis is expressed in a quantitative way as well and it considers the calculated fitness of the motion and not actual realism.

In order to have realistic results, the optimization has to be led to the desired natural-looking solution. Liu and Popovic [2002] stated that this would happen with proper goals that encourage natural motion. Minimizing the spent muscle energy, as described in Chapter 5.2, is an example of a human-like behavior that is attempted to be simulated. However, there are many

cases where minimizing the energy does not lead to the desired motion, such as depicting emotions like being excited, as explained by Geijtenbeek and Overmars [2011]. Liu and Popovic [2002] stated that even realistic results may also not be artistically pleasing. Rose et al. [1998] also found out that working with high-level control parameters can be unintuitive, as explained in the earlier chapter, and controlling the motion by tuning parameters is generally different from how traditional animators are used to work. The following sections explain why creating natural and artistic motion with trajectory optimization is difficult and what kind of solutions have been tried earlier.

6.1 Under-Constrained Trajectories

Many attempts to create plausible motion with optimization techniques, such as by Popović and Witkin [1999] and Hodgins and Wooten [1998], focus on high-energy motion like athletics. That kind of highly dynamic motion is mostly constrained by the physics-based equations of motion and does not allow for a large stylistic expression. The less constrained the motion is, the larger is the space of possible motions from which to pick the right one and the more room there is for emotional or artistic expression. Walking, running and manipulating objects are examples of loosely-constrained motion, as explained by Hodgins and Wooten [1998]. These stylistic variations could be viewed as adjectives describing the style of the motion. As an example, walking could be lazy, energetic, exhausted or sneaky. All these variations still bring the character forward. In contrast there may only be one way for the simulated character to do a proper gymnastic vault, and there may be no way to do e.g. a sneaky or an exhausted vault.

Controlling the stylistic variation would require sophisticated goal formulations to lead the optimizer to the desired solution. The goals may be generic and always present or added when the solution looks wrong as done by Yin et al. [2008]. Additional goals and constraints that would lead the optimization to human-like motion, as the earlier mentioned goal of energy minimization, have therefore been experimented since the invention of space-time optimization, but, as shown by Al Borno et al. [2013], proven to be difficult. One reason is that a realistic models for muscle actuation and the elastic properties of the tissue are not generally used. Even if energy minimization could be perfectly implemented, even human walking is shown not to use muscle energy optimally, especially as Carrier et al. [2011] explained that the energy efficiency depends on the speed of the locomotion. And even if the goal formulations would be perfect, the under-constrained trajectories

lead to a large space of motion candidates to consider. That, combined with the large number of degrees-of-freedom in human models, makes it difficult for optimization methods to find the perfect solution. This problem is further detailed in Chapter 7.

6.2 Modelling Realistic Motion

So far, as stated by Al Borno et al. [2013], optimization methods have not been able to synthesize motion which appears perfectly natural. This is due to the difficulty to model the aspects of human motion that make it natural and to formulate parameters which describe the style of the motion. Liu et al. [2005] also explained how the human model also needs to be realistic enough to be able to reproduce natural motion. On the other hand, the style of the motion has also not been the main focus in many studies, such as by Mordatch et al. [2012] and Hodgins and Wooten [1998]. Recently there have been several attempts for data-driven styling, for example by Liu et al. [2005]. This is most commonly done by using deriving styling aspects from motion capture data, as done by Da Silva et al. [2008]. The motion capture based approaches try to separate the styling data, such as happiness, from the basic motion e.g. walking. Rose et al. [1998] called this separating the verbs describing the basic motion from the adverbs which represent the style of the motion.

The method Liu et al. [2005] used was to compare the animation produced by motion capture with synthesized animation and assumed that the differences between those two animations are explained by the style of the captured motion. Calculating the difference requires that the optimization produces similar motion as the captured motion, which may not happen due to the differences between the real and the simulated human model. The method would also not be optimal for processing hand-made animations because while captured motion is physically correct, keyframe animation in general does not have to be. Also, even if the body movements caused by styling can be detected, the style information has to be interpreted in a higher level so that it can be applied to a different motion. Even with reference data, Safonova et al. [2004] stated that the natural aspects of the motion are still hard to describe with mathematical models. Liu et al. [2005] attempted that by capturing the style as the relative preference to use certain muscles over others and as the elasticity of the muscles, tendons and the shoes that the person was wearing. They also claim that their method does not require a large set of example motions to learn the styling data from, which would otherwise be an issue in the case of this thesis, since the aim is to minimize

the amount of motion capture data that is required to develop a game.

To reduce the amount of learning that is needed the styling parameters could be interpolated to create a new set of parameters which combine several emotions or display them in varying levels. Rose et al. [1998] approached this by defining a linear parameter for each emotion, such as one defining how sad or happy the person appeared to be. The level of happiness of the character could be gradually changed from sad to happy by increasing the value of the parameter, allowing for a simple high-level control over the desired emotional state. This kind of linear interpolation requires that the style of the motion could be expressed with a parameter space where the basis vectors are formed from the opposite emotions. As done by Rose et al. [1998], the internal model of the motion styling for different emotions can again be derived from motion capture data, where the captured motion expresses a pre-defined set of emotions. For example, the happiness parameter is constructed so that it explains the main differences between happy and angry motion clips. Often, as in the work by Liu et al. [2005], the style is assumed to be invariant during the motion. However, that would not allow for generating motion which blends between different emotional states. By allowing the styling parameters to gradually change during the generated motion, such transitions could also be made.

Even if the style of the motion would be perfect, the overall control decisions of the simulated character can still feel unnatural. Humans are not perfect with their actions. They make mistakes, may not be able to see or otherwise sense everything and take time to act to a sudden change in the situation, such as a loud noise coming from behind. On the other hand a computer simulation, especially when not limited by processing time, may end up with a response to an external stimulus that appears too perfect to be natural, leading to seemingly super-human reactions. Zordan et al. [2007] proposed to alleviate this by adding delays to response times, simulating the way it also takes time for humans to react to sudden events. They also mentioned using random noise in the motion and letting the simulated humans to fail some of their actions. Al Borno et al. [2013] proposed parameters which would allow the adjustment of how super-human the motion produced by the optimization should appear to be. Jain et al. [2009] also implemented virtual sensors which the character uses to gather information about its surroundings, for example if there is a handrail nearby to grasp with its hands to provide help with balancing.

Chapter 7

Optimization

Spacetime optimization attempts to find the trajectory which provides the maximum fitness value, as this fitness value tries to describe the quality of the resulting motion. The fitness function, introduced in Chapter 5, is constructed from the defined goals, and is a function which takes the trajectory control parameters as its arguments and outputs the fitness value. The maximization problem (Formula 7.1) then tries to find the trajectory parameters x for which the fitness function f provides the highest value. The feasible region $\mathbb{C}$ of the maximization, the region of parameter space to be searched, is defined by the constraints of the optimization problem.

$$\vec{x}_{best} = \arg\max_{x \in \mathbb{C}} f(\vec{x}) \tag{7.1}$$

As explained in Chapter 5.3, the number of optimized parameters becomes large with high-DOF models, long spacetime optimization window and several optimized control points. Therefore the parameter space has also equally large number of dimensions. As noticed by e.g. Muico et al. [2009], this high dimensionality makes solving the optimization problem difficult and time-consuming. Mordatch et al. [2012] explained how the problem is made even more difficult by the discontinuous and non-linear feasible regions. These difficulties and the ways that have been tried to overcome them are detailed in the following sections.

7.1 Problem Landscape

The high dimensionality of the search space means that due to the curse of dimensionality its volume is also huge and therefore it would take a long time to explore exhaustively to find the point of maximum fitness, as shown by

Kuo and Sloan [2005]. But that is not the only issue with the optimization landscape. Another issue rises from the way how rigid bodies in a physical simulation are commonly prevented from overlapping each other. Especially ground contacts such as with the ground are necessary for the character to move due to being under actuated. As explained by Muico et al. [2009], because the bodies may not move so that they overlap, and since multiple different kinds of trajectories may satisfy the goals, the parameter space becomes discontinuous, having multiple separated feasible regions that need to be searched. Being discontinuous also guarantees that the parameter space has multiple local maxima, where the fitness function has a higher value than the region surrounding it. This is because each separate feasible region has at least one local maximum inside of it. If the used optimization method is unimodal, meaning that it converges to the nearest local maximum, it is likely to be unable to find the global maximum, which would be the best solution. There have been attempts to formulate the parameter space so that it has better characteristics. Mordatch et al. [2012] used a method which avoided discontinuities in the parameter space, and Tassa et al. [2012] attempted to smooth out the sharp edges of the feasible regions.

There may be too many constraints for any solution to be possible. An example of this would be if the start and the end positions of the character are fixed as constraints, but there is no way the character could reach the destination, e.g. when it is in a different side of a wall than the starting position. This means that the optimization problem is over-constrained and cannot lead to a solution. The feasible region has then a zero size. Liu and Popovic [2002] solved these cases by loosening the physical constraints, reducing the realism of the simulation but at least yielding a solution.

7.2 Gradient-Based Optimization Methods

Since a closed-form solution to the spacetime optimization problem is generally not found, the fitness function is commonly treated as a black box, in other words that its implementation details are not considered as to be known during the optimization. Iterative optimization methods then use the fitness function to sample a sequence of points in the parameter space, and using the results, produce approximations for the solution which are increasingly better. Witkin and Kass [1988] proposed sequential quadratic programming (SQP) for iteratively solving the spacetime optimization problem. In quadratic programming, the problem is formulated as a set of equality and inequality constraints which must be satisfied and a minimized objective function, which could be viewed as a reciprocal of the fitness function.

Quadratic programming-based solutions work most efficiently when the dimensionality of the parameter space is as small as possible, which is commonly not the case when constructing human motion trajectories. One issue with SQP is that it requires a costly inversion of a matrix the size of which depends on the number of optimized variables and therefore becomes much slower when the dimension of the parameter space is large.

Another issue is that the method requires the Jacobians of the constraint functions and the Hessians of the fitness function, for which there may not be an analytical solution. Calculating them with finite differencing, by using the difference of the fitness value of a searched point in the fitness landscape and its close neighbors, is slow, again due to the large dimensionality of the parameter space. Tassa et al. [2012] noticed that nearly all the processing power spent by the optimization was consumed by calculating the finite differences. To improve performance, Yin et al. [2008] searched a limited number of points around the current sample in a random fashion to reduce the amount of calculations required to find a suitable direction to step towards. They noticed that convergence this was was much faster but did not end up with a solution that was as good as with the other methods they tried which were based on a linear search. The performance can also be affected by the stepping method. Besides the Newton-Raphson step used by Witkin and Kass [1988], various alternatives exist, such as the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) used by Rose et al. [1996] and Mordatch et al. [2012].

7.3 Stochastic Optimization Methods

Gradient-based methods are usually greedy, always stepping towards a better fitness value. Therefore they are prone to getting stuck into local minima, since they generally cannot step into a point in the optimization space with a worse fitness value and that way get out of the local minimum. This limitation can be overcome by using stochastic methods, which use a pseudo-random process to find the global maximum. A wide variety of stochastic optimization methods exist, such as simulated annealing, genetic algorithms and particle swarm optimization (PSO). Closest to gradient-based methods is perhaps simulated annealing, introduced by Laarhoven and Aarts [1987], in which the optimization has a chance of stepping towards a state with a worse fitness value instead of always stepping to a point with a higher fitness. This chance may be affected by how much worse the candidate state is than the currently considered state and the chance to step towards a worse state may also diminish over time, gradually increasing the greediness of the

algorithm. However, due to its nature of not keeping track of any past states, the algorithm may have to go through the same areas of space multiple times. As always, the large dimensionality slows down the algorithm considerably as well.

Many of the stochastic methods keep track of multiple locations in the parameter space at once to explore multiple promising areas at the same time. This way the algorithm will not easily get stuck in just one favourable area. Genetic algorithms are an example of searching multiple locations. They are inspired by the natural evolution, which randomly mutates and permutates the genes of the offspring. According to the theory of natural selection, the best genetic mutations give the offspring a competitive advantage in their life, increasing their chances of breeding. Inspired by the nature, the genetic algorithms track multiple samples, which on each iteration generate offspring. The offspring have some of their parameters inherited from that parent sample and other parameters from an another parent sample. Then the parameters are randomly mutated. After that, the population is culled so that the samples with the highest fitness have a better chance of surviving. The intuition of the algorithm is that finding a sample with a high fitness means that there are likely even better samples to be found nearby, and also that the exchanging of parameter values when generating the offspring could in the best case combine the best features of the parent samples. [Holland, 1992]

On the other hand, in particle swarm optimization, the population does not change between iterations, but instead is formed of particles which fly around the optimization space. Each particle samples its location after each iteration and remembers the best location they have found so far. They also know the best location that any of the particles has found. During each iteration, a force pulls the particles towards the best location they have found themselves, as well as the globally best location that is found. Because the particles have a velocity, they will often overshoot those locations of high fitness while flying and that way may randomly end up to an even better position, which they will then remember instead. [Kennedy et al., 1995]

Instead of keeping track of a list of locations, another way is to form an estimate of the fitness landscape based on the already evaluated samples and iteratively improving this estimation after every drawn sample. Hämäläinen et al. [2006] modelled the fitness function landscape as a mixture of Gaussians. The model is then used as an unnormalized probability density for drawing samples, concentrating the sampling to areas which seem the most promising so far. After the fitness of each generated sample is evaluated, the estimation of the fitness landscape improves. Since the unexplored areas could have maxima which are not known yet, the method also samples areas

which have not seem very promising so far if they are large enough.

7.4 Initial Guess

Optimization methods usually benefit largely of a prior estimation of a good sample, which in this case would be a trajectory which is as close to the optimal solution as possible. This initial guess, if close to the global maximum of the fitness landscape, often helps the optimizer to quickly converge towards that maximum. For example, gradient based methods, when starting close enough to the global maximum, could climb a hill leading to the global maximum to quickly reach it. Similarly stochastic methods could start with a better estimation of the fitness landscape if given a few decent samples to begin with and therefore could draw the new samples from better places. On the other hand, an initial guess around a sub-optimal local maximum may end up with the optimizer, especially gradient-based methods, getting stuck in that local maximum. Even if a stochastic method would not get completely stuck, it could unnecessarily spend samples for investigating the space around the local maximum.

7.5 Simplifications

In order to overcome the previously presented difficulties in spacetime optimization, several kinds of simplifications to the optimization problem formulation have been proposed earlier. Safonova et al. [2004] and Popović and Witkin [1999] claimed that many typical human motion trajectories could be expressed with much lower actual degrees-of-freedom (DOF). This is e.g. due to the symmetry of left and right sides of the body and because some muscles are not relevant for the desired motion. Because of symmetry, the rotations of the joints of both left and right legs could be locked to be the same for motions such as jumping. Reducing the DOF also lowers the dimensionality of the landscape, mitigating the issues caused by the high dimensionality. However, Hodgins and Wooten [1998] stated that there has to be enough DOF available in the character model for the motion to look realistic, which may not happen in simplified models such as if the left and right sides of the body are mirrored. The required DOF also depend on the type of motion, which is not known beforehand in this thesis, making it unsuitable to lower the DOF.

Al Borno et al. [2013] reduced the number of optimized variables by not considering the entire planning horizon at once. Instead they split the horizon

to shorter, half a second time windows and considered only two subsequent windows at once during the optimization. After each two-window part of the horizon was optimized, the results regarding the second window were discarded and then the optimizer processed the second window again together with the window which was right after it in the timeline. However, limiting the optimization effectively to one second planning horizon at a time still limits its way to anticipate events further away in time. It also required that the goals were formulated for each window of time separately so that they could actually be evaluated for a partial planning horizon. This means that a goal for having the desired end pose is not enough, but intermediate goals such as poses leading to the end pose are needed as well. Therefore such a method is not suitable in this thesis as the minimal amount of user input should be required and specifying the intermediate poses would be difficult and time-consuming.

The difficulty of the optimized task can also be gradually increased by defining a variable describing the gradual difficulty increasing. Mordatch et al. [2012] implemented physics as goals instead of constraints, gradually increasing the fitness cost of violating physical constraints. They also used temporary goals to guide the optimization to the right direction, and removed those goals before the optimization was finished. Yin et al. [2008] adapted a walking controller to climb stairs, gradually increasing the height of the step. Panne and Lamouret [1995] used auxiliary, "hand of God" forces to guide the movement, gradually diminishing the guiding force. Another method that they described was to help balancing by starting with huge feet, which would have a large surface area colliding with the ground, preventing the character from tripping. The size of the feet would then be gradually decreased during the optimization process. Similar kind of gradual improvement of motion is attempted at this thesis by allowing the gradual changing the goal parameters.

Chapter 8

Spacetime Optimization Formulation

The goal of the thesis was to try out how well a stochastic spacetime optimization method can generate various kinds of motions with minimal tuning of the optimization to the specific motion. As explained in the earlier chapters, spacetime optimization has been a promising method for generating a wide range of motions, but the existing methods still suffer from the difficulties of the optimization problem. Using a stochastic optimization method could offer a solution to many of the issues caused by the shape of the optimization landscape, as explained in the previous chapter. In this thesis we have used a stochastic spacetime optimization method which is based on importance sampling. The following sections explain how the spacetime optimization problem was formulated in this thesis.

8.1 Trajectory Formulation

As detailed in chapter 5.3, using control points for trajectory formulation requires less optimized parameters than specifying joint angles for every instance of time inside the planning horizon. The more control points there would be, the more detail the movement could possibly have, but the longer the optimization would likely take due to the increased amount of optimized parameters. In our solution the number of control points had to be predefined by the user. In contrast to the work by Al Borno et al. [2013], the control points do not have to be evenly spaced, but instead the position of the control points in the timeline were implemented as optimized variables. This way the optimizer could place the control points freely to any point in the timeline where they would be the most useful, for example if the motion

would have some key postures which describe it such as first crouching and then jumping. The optimizer could get rid of unnecessary control points by specifying their duration to be zero, but it could not add more control points if a motion requires more than the pre-defined maximum amount. The maximum duration between control points was defined to be 0.5 seconds, which also automatically limited the maximum duration of the simulation.

Besides timing, the control points contain target values for the joint angles which the character attempts to reach at the specified time. The target angles were also be forced to stay inside the defined angular limits of the joints, which makes the optimized parameter space smaller. The control points also include variables describing the maximum values for the joint torques that different parts of the body can use. Having a separate torque limit for each joint would drastically increase the amount of optimized parameters. Instead, the body parts are organized to 3 groups regarding the torque limits, having the same limit for all the joints in the arms, one shared limit for the joints in the legs and a third limit used by every other body part (Figure 9.1). This was done with the anticipation that due to the symmetry of various kinds of movements, such as walking, both sides of the body would likely use the same stiffness for the muscles for both the mirrored body parts such as legs. It was also assumed that the stiffness of all the muscles in each major body part such as a hand or a leg could be treated as being the same to further reduce the number of optimized parameters. The joint torques were limit to 100Nm.

The control parameters outside the control points were interpolated using cubic splines. Using splines instead of a linear interpolation of control points was expected to make the motion smoother and continuous, and was also supported by the earlier works such as by Al Borno et al. [2013], as explained in Chapter 5.3. The same kind of interpolation was used for both the target joint angles and parameters describing the maximum forces. During the physical simulation, after each simulation time step, the control parameters would be calculated based on the current point of time and the values of the control points. The physical simulation would then be instructed to drive the joint motors to the calculated target angles while respecting the calculated maximum torques for the motors.

8.2 Constraints

All physical constraints were enforced by the physical simulation itself. This also included the angular limits for the character's joints. The only constraint besides the parameter limits was that the starting state of the character was

constrained to be the same as the last pose of the source animation. The starting state contained the position and the rotation of the root of the character, the angles for each of the joints and the angular and linear velocities for every rigid body. Obviously the position and the rotation of the character should also be the same at the start of the transition animation as they were at the end of the source animation. In contrast to many earlier works such as by Al Borno et al. [2013], who assumed that the character started stationary, the velocities were defined for the starting pose as well. This was necessary because the character was supposed to transition from one motion to another. As an example, when transitioning from a running animation to a crouching animation, the root velocity at the end of the running animation had to be taken in account. Also to avoid sudden discontinuities in the trajectories of the body parts, their linear and angular velocities should also be continuous at the point when the transition animation starts. Since the source animation was assumed to be a plain keyframe animation without any metadata information like body velocities, the velocities had to be calculated using a backward difference. The position and rotational angle of each body part were calculated at the end of the source animation and also at 0.2 seconds before the end.

The linear velocities were calculated from the positional backward differences (Formula 8.1):

$$\vec{v}^{p}_{end} = \frac{\vec{P}(p, t_{end}) - \vec{P}(p, t_{end} - \Delta t)}{\Delta t} \tag{8.1}$$

In equation 8.1, p is the index of the body part, v^p_{end} is the calculated starting velocity of a body part p at the selected transition time t_{end}, $P(p,t)$ returns the position of the body part p at the time t and Δt is the time interval for calculating the positional differences ($\Delta t = 0.2s$).

Likewise the angular velocities Ω^p_{end} were calculated from rotational differences (equations 8.2 and 8.3):

$$\Delta r^{p}_{end} = R(p, t_{end}) * R(p, t_{end} - \Delta t)^{-1} \tag{8.2}$$

$$\Omega^{p}_{end} = Ang(\Delta r^{end}_{p})/\Delta t \tag{8.3}$$

In the equations, $*$ denotes quaternion multiplication, $R(p,t)$ returns the rotation of the body part p at the time t. The notation r^{-1} denotes the inverse of the quaternion r. The function Ang calculates the axis-angle representation of the rotation and returns the angle of the rotation. The rotational axes are defined by the joints.

8.3 Goals

The transition animation should end in such a way that it is directly continued by the target animation. However, as explained in Chapter 5.1, the end pose may not be able to be reached exactly, so defining it as a constraint could be an issue. Also due to the decision of not using a quadratic programming formulation for the solution and the way how the system was built using a stochastic optimizer and a physical simulation, the end pose was also impossible to be specified using a constraint. Therefore it was specified as a goal instead. The target end pose was defined in the same way as the starting pose, containing both positions and velocities for the bodies. A goal was formulated to determine a fitness value based on the closeness of the root position, body positions and linear velocities to the target positions and velocities. Angular velocities were not used in the goal formulation.

The entire movement trajectory of the character was searched for a pose which best matched the target end pose by calculating the fitness value of the goal after every evaluated physical simulation step (frame). The point of time when the character best matched the end pose was determined and the generated trajectory was immediately cut at that point of time, completely discarding the rest of the trajectory after it. This way the optimizer did not have to find a trajectory which ended at the desired pose but could instead find one which passed through the pose at any point of time. This was assumed to speed up the optimization process as the number of possible solutions was larger. On the other hand, the control points after the determined end pose will likely not affect this cut trajectory a lot and could confuse the optimizer as they would have very little effect to the resulting fitness value of the trajectory. Regardless, the initial tests indicated that ending the trajectory prematurely improved convergence times. Since the end pose would likely not be reached exactly, when exporting the simulation results as an animation the ending of the animation was were linearly interpolated with the end pose so that the animation would end up exactly at the desired pose.

The fitness value of every goal was evaluated by first calculating the error values, the differences between the desired and realized values, such as the desired and actual joint angles. The priority of a goal was defined by dividing the error with a scaling multiplier σ_g, where g denotes the goal. Since the priority values have to be hand-picked, determining the best values requires some level of experience from the user. However, in a similar case, Al Borno et al. [2013] stated that the goal weights do not need to be exactly perfect to produce a desired animation.

The pose goal, which evaluates how well the end of the trajectory matches

the end pose, calculated the positioning error with Formula 8.4 for the joints and with Formula 8.5 for the root location in each frame i.

$$\mu_P^{(i)} = \frac{1}{\sum_{j=1}^{N_j} k_j} \sum_{j=1}^{N_j} k_j \|\vec{p}_i^{(j)} - \vec{e}_i^{(j)}\|^2 \tag{8.4}$$

$$\mu_R^{(i)} = \|\vec{r}_i - \vec{g}_i\|^2 \tag{8.5}$$

In these formulas $\vec{p_i^{(j)}}$ is the location of a joint j during the frame i, $\vec{e_i^{(j)}}$ is the target location of that joint and k_j is the importance of the joint, which is 2 for the arms, legs and head and 1 for the other body parts. $\vec{r_i}$ is the location of the root of the character in the frame i and $\vec{g_i}$ is its target location. The joint velocity error $\mu_V^{(i)}$ was calculated in the same way as the joint positioning error, with velocities substituted for positions. The fitness value of the pose goal $f_E^{(i)}$ was computed for every frame with the equation 8.6 and the spline is cut at the frame i_{max} where the pose goal yields the highest fitness f_{Emax} (equations 8.7 and 8.8).

$$f_E^{(i)} = e^{-\frac{1}{2}\left(\frac{\mu_P^{(i)}}{\sigma_P^2} + \frac{\mu_R^{(i)}}{\sigma_R^2} + \frac{\mu_V^{(i)}}{\sigma_V^2}\right)} \tag{8.6}$$

$$i_{max} = \arg\max_i f_E^{(i)} \tag{8.7}$$

$$f_{Emax} = f_E^{(i_{max})} \tag{8.8}$$

The rest of the goals would use the truncated trajectory determined by the pose goal instead of the full trajectory. Those goals were minimizing the jerk of the body movements and avoiding collisions for vital body parts, both which were inspired by earlier works as detailed in Chapter 5.2. Both acceleration and torque minimization were also tried out, but neither of them yielded as visually pleasing results as jerk minimization. Jerk minimization also considered the end of the source animation and the start of the target animation to ensure that the motion stays smooth when the animation of the character changes instantly. The state of the source animation 0.1 seconds before the end was used as an evaluated point of time in the jerk calculation, placed right before the physically evaluated time frames. Similarly the start of the target animation was taken in account as well, evaluating the animation 0.1 seconds after the start and considering it as a frame after the last physically evaluated time frame. At each simulated time step, the amount of jerk $\dddot{\vec{p}}_i$ was calculated with the equation 8.9. The goal was formulated

to minimize the mean squared jerk, which was computed with the Formula 8.10.

$$\dddot{\vec{p}}_i = \frac{\vec{v}_{i-1} - 2\vec{v}_i + \vec{v}_{i+1}}{(\Delta t)^2} \tag{8.9}$$

$$\mu_J = \frac{1}{N_f} \sum_{i=1}^{N_f} |\dddot{\vec{p}}_i|^2 \tag{8.10}$$

Collisions to vital body parts, which were considered to be the head and the chest of the character, were also discouraged by determining an error value. The error would be zero if the body part does not collide with anything during the motion and defined to be non-zero if the body part collided even shortly. The error value would be the same regardless of the duration or the count of the collisions. The error value for the collision avoidance goal μ_D was determined by the Formulas 8.11, 8.12 and 8.13. There the both body parts had their own multipliers for the error value, which were m_h for the head and m_c for the chest.

$$\mu_D = e_h + e_c \tag{8.11}$$

$$e_h = \begin{cases} m_h, & \text{if the head has hit an object before the frame } i_{max} \\ 0, & \text{otherwise} \end{cases} \tag{8.12}$$

$$e_c = \begin{cases} m_c, & \text{if the chest has hit an object before the frame } i_{max} \\ 0, & \text{otherwise} \end{cases} \tag{8.13}$$

Due to the piecewise functions for the error values, such a goal would be especially unsuitable for gradient-based optimization methods, but could also be problematic for the stochastic optimization method which was used. An another way would have been to penalize the collision velocity, having a higher error value with a higher collision velocity. However, this would not penalize motions which would carefully place the head on the ground first. It would have also been possible to penalize collision forces, but that could have allowed the character to slide its head gently along the ground.

The final, combined fitness value of all the goals was determined by:

$$f = f_{Emax} e^{-\frac{1}{2}\left(\frac{\mu_J}{\sigma_J^2} + \frac{\mu_D}{\sigma_D^2}\right)} \tag{8.14}$$

Other goals such as balancing goals, introduced in Chapter 5.2, were tried as well, but were not determined to be necessary. If the character would lose balance, it would likely not regain it during the limited time frame allocated for the transition animation, especially as the jerk minimization goal was also present. Therefore it would not likely reach the desired end pose as well. Similar to Al Borno et al. [2013], we also noticed that using as few goals as possible gave better results and made it easier to adjust the goal parameters and generalized the goals better for different kinds of tasks. As an example, determining the best kind of a goal for balancing, which would suit all kinds of motions, is difficult. Walking and running require the character to be slightly out of balance at all times while standing does not.

8.4 Initial Guess

As explained in Chapter 7.3, a stochastic optimizer benefits a lot of a decent initial guess for the trajectory. In other words, initializing the optimizer with a set of trajectory parameter values, which are as close to the optimal values as possible, improves convergence times a lot. To generate a decent initial guess, we linearly interpolated the parameter values between the start and the end poses, like Safonova et al. [2004] did, placing each control point evenly in the timeline so that their distances were half of the defined maximum distance. Even if the linear interpolation was simple and would rarely result in the best possible motion, we assumed that it could capture some essential, overall characteristics of the transition, such as the movement of the COM in a crouching motion.

In addition, in order to further encourage a smooth transition from the source animation to the transition animation and from the transition animation to the target animation, the second and second last control points were defined the initially guessed trajectory so that they would continue the motion of the source and target animations for 0.1 seconds. The control points were calculated assuming that the velocities of the body parts would not change during 0.1 seconds after the source motion and 0.1 seconds before the target motion. When the optimization started, the optimizer was first instructed to evaluate the guessed trajectory. If its fitness value would be decent, the optimizer would likely search the parameter space around it for even better solutions, quickly giving some fairly good results.

8.5 Optimization

Spacetime optimization was implemented using a ready-made implementation of mutated kD-tree importance sampling, which was first introduced by Hämäläinen et al. [2006]. The method was further improved during the development of this thesis. The largest improvement was the implementation of a sequential sampling mode, which uses the results of a prior optimization task to speed up the convergence time in a similar, subsequential optimization problem. The implementation of the sampler was not part of this thesis, and it is therefore treated as a black box.

Chapter 9

Implementation

9.1 Physical Simulation

The physical simulation was treated as a black box during the optimization, so the optimizer had no knowledge of its internal operation. Actually, the optimizer only produces trajectory parameters and receives back the fitness value of the resulting trajectory, without even knowing how the trajectory parameters are used to produce the fitness value. The trajectory is calculated as detailed in Chapter 8.1, by running the physical simulation step-by-step and storing the results. Due to the loose coupling of the optimizer and the physical simulation, any deterministic off-the-shelf physical simulator library should have worked with the system. This means that if the produced animation is used in a game, the same physical simulator could be used for producing the transition animation and for running any physical simulations that the game requires. This could be helpful, since the environment where the character acts would need to be built only once. Similarly, Stewart and Cremer [1992] also used a general purpose physical simulation library, but they had also built a custom interface for the motion synthesizer to communicate with the simulator.

The simulator needed to be deterministic, however, so that if a trajectory would be evaluated twice the results would be the same both times. Any randomness in the simulation would have confused the optimizer which expects that the mapping between the trajectory parameters and the fitness values remains unchanged. The luck factor of a nondeterministic simulation could lead the optimizer to make wrong conclusions. However, physics simulators commonly used by games are generally nondeterministic for performance reasons. The same was true in Open Dynamics Engine (ODE), the physics simulator we ended up using. Since computer-generated pseudo-

random numbers are generally seed-dependent, as was the case with ODE, storing the seed value and then using the same seed for all simulations made the simulation deterministic regarding to random numbers. Removing certain optimizations from ODE such as reordering of internal arrays was also needed for completely deterministic operation. The iterative linear complimentary problem (LCP) solver in ODE was found to be unstable unless the length of a simulation time step was very small, which was unpractical regarding to other calculations such as goal evaluation. Therefore the direct LCP solver of ODE was used instead, due to being much more stable in larger time steps. We also modified ODE so that it was completely causal and so that any occurring errors would not cause the application to crash but instead stop the simulation and discard the evaluated trajectory. Code was also added to enable saving and loading the state of the physics engine.

9.2 Character Model

In order to be able to produce a large range of different movement trajectories, the character model which we used consisted of 30 actuated DOF. The model does not contain any finer details such as fingers or toes. The expectation was that they are not necessary as transition animations are generally very coarse-grained and do not have such a detailed motion as grabbing an object, which would have required modelling the fingers in some detail. Instead, hands and feet were modelled with a single rigid body for each of them. This saved us from modelling at least one universal joint and 2 hinge joints per finger, giving a total saving of 40 unnecessary DOF that were removed from fingers alone, simplifying the model considerably. The same simplification was made universally in all the earlier works which synthesized full-body motion as well. A further simplification we made was the omission of sternoclavicular joints which would have allowed the rotation of clavicles. This makes the upper torso appear stiffer, but the issue is mitigated because the torso is split to upper and lower parts, letting the upper part to rotate in relation to the lower part.

Every other joint was modelled with a ball-and-socked joint except for elbows, knees and ankles, which were modelled using hinge joints. For elbows and knees that is natural, but modelling the ankles with hinge joints means that they can only be flexed but not rotated. This limitation is somewhat alleviated by hip rotation and because ankle rotation was not considered to be important for most simulated motions. In order to save computing power, collisions between different parts of the body were not considered, as has also been done by Mordatch et al. [2010] and Wu and Popovic [2010]. Instead the

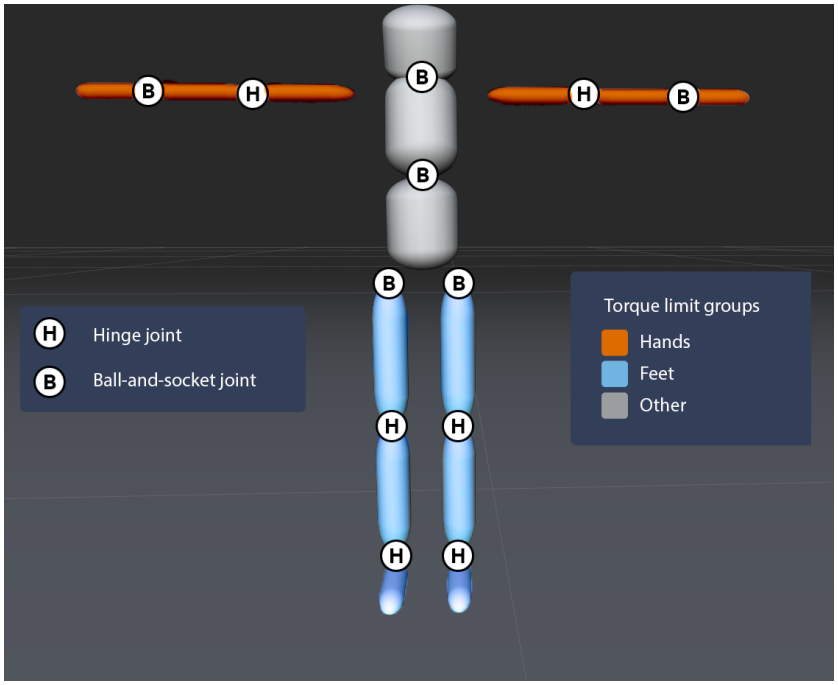

Figure 9.1: The implemented character model in the T-pose. The size of each body part was chosen to approximate its relative weight.

maximum angle of each joint was overly limited to prevent any possibility for the body parts colliding. As explained in Chapter 5.2, this naturally limits the possible trajectories that can be synthesized. Besides saving computation time, it also does not require special handling of the collisions between the body parts which are connected directly by a joint. Implementing collisions between body parts is therefore left as a future work.

9.3 Parallelization

Modern computers have several processor cores, so using only one of them for the computationally-intensive optimization task would have been a waste of processing power. Therefore Al Borno et al. [2013] and Tassa et al. [2012] parallelized their optimizers to utilize all the processing power. Also in our case we implemented the optimization process so that it was possible to be highly parallelized. The pseudo-code for the threaded sample evaluation is presented by Algorithm 1. First, before the actual optimization process started, the starting state of the physical simulation was calculated only once

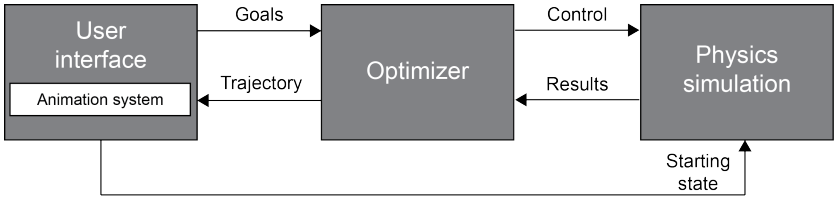

Figure 9.2: System architecture and its main high-level components.

and then stored for later use. One thread was then started for each logical core in the processor. Each of the threads would initialize their own physical simulation according to the previously stored starting state.

After that a thread would request a sample from the optimizer and simulate it in its own instance of the physical simulation. Then the fitness value of the sample was calculated in the same thread and delivered back to the optimizer. If the sample was better than any earlier evaluated samples, the results of the physical simulation were stored in the animation display system. After the sample evaluation and animation generation, the thread would reset the physical simulation and re-initialized it to the starting state and the sample evaluation would happen again. This procedure would be continued in a program loop until the user would stop the optimization process. When the animation display system had finished playing the previously generated animation, it would remove any previously stored simulation results from the store if there was any. Then the animation display system would generate a new animation based on the simulation results and replace the previously displayed animation with the newly generated one. Algorithm 2 contains the pseudo-code for the animation display system.

Access to the optimizer and to the animation storage of the display system were not thread safe, so in both cases the access to the optimizer or the animation storage was limited to only one thread at a time. This was done by using the native C# monitors, which are explained in more detail by Stärk [2005]. The monitors do not ensure that the threads attempting to enter it would be served in a first in, first out (FIFO) order. Therefore, after the simulation thread entered the animation display monitor, the thread first checked if any other thread had stored even better results while the thread was waiting for the monitor. In that case, the thread would not store its results to the animation display system, as better results would have already been found by an another thread.

Algorithm 1 Threaded Sample Evaluation

```
storedPhysicsState ← physicsState
bestFitness ← 0
displayedResults ← empty
for all logical cores in processor do
    in new thread
        while not stopped by user do
            enter monitor optimizerMonitor
                sample ← GenerateSample()
            exit monitor
            threadPhysicsState ← storedPhysicsState
            simulationResults ← SimulatePhysics(sample, threadPhysicsState)
            sampleFitness ← CalculateFitness(simulationResults)
            enter monitor optimizerMonitor
                NotifyOptimizerOfFitness(sample, sampleFitness)
            exit monitor
            if sampleFitness > bestFitness then
                enter monitor animationDisplayMonitor
                    if sampleFitness > bestFitness then
                        bestFitness ← sampleFitness
                        displayedResults ← simulationResults
                    end if
                exit monitor
            end if
        end while
    end thread
end for
```

Algorithm 2 Animation Display System

```
▷ displayedResults is defined in Algorithm 1
animation ← empty
while not stopped by user do
    PlayAnimation(animation)
    enter monitor animationDisplayMonitor
        if displayedResults ≠ empty then
            animation ← GenerateAnimation(displayedResults)
            displayedResults ← empty
        end if
    exit monitor
end while
```

9.4 User Interface

The user interface allows the user to specify at what point in the source animation the transition should start and at what point in the target animation the transition should end up to. The start and end poses would then be calculated, also containing the velocities for each of the body parts. The user interface allows for fine-tuning the postures by rotating and moving the body parts, but not changing the velocities and torques of the parts. The poses and animations were displayed in two separate views, one viewing the character from the side and one viewing from the front. Linear velocities of the rigid bodies were indicated by lines starting from the center of the rigid body, having the direction of the motion and the length in proportion with the speed of the movement. Torques were similarly indicated with differently colored lines, the direction of the line indicating the axis of rotation and the length of the line being in proportion to the rotation speed. Another kind of indication of the movement was displayed by having ghost characters to indicate the animation frames 0.1 seconds before the starting pose and 0.1 seconds after the ending pose. The poses could be saved to a file and loaded back later. The resulting animation could also be saved, though only in the Legacy animation format in Unity, which is discouraged for character animation. This was due to the limitations of the programming interface in Unity.

Sliders were presented to modify the changeable parameters of the goals. Both slider values and poses could be changed also during the optimization, which would force the optimizer to re-evaluate any already evaluated samples. During the optimization, the error values of each of the goals were displayed for the currently best sample. The full motion of the currently best sample would be played so that the user could see if the animation is suitable for their needs. The playback started with the source animation, then played the transition animation and finally ended with the target animation. An option to also display only the collision geometry instead of an animated human character was also presented, to see if there were any issues with the simplified fixtures of the character. A graph would also display the development of the best fitness value found so far. This made it possible to see if the fitness had not increased significantly for a long time, which indicated that either a better solution does not exist or it is difficult for the optimizer to find.

Chapter 10

Experiments

The transition generation was supposed to allow for fast iterative tuning of the goal parameters of the trajectory. Similar to traditional digital animator work, as explained by Parent [2002], tuning the control parameters of the animation was still expected to require trial-and-error work. After all, expressing the system what kind of an animation is preferred by using goal parameters is not simple, and likely the animator could only know when the animation is correct when they actually see it. To aid this iteration the stochastic optimizer, which is used in this work, allows for using the previous optimization results to speed up the following optimization run. This was expected to work even if the goal parameters were changed, due to the decision of using a novel sequential kD-tree-based importance sampling method, described in Hämäläinen et al. [2006]. The sequential sampler uses the estimation of the fitness landscape from the previous optimization task to help the convergence of a subsequential optimization task. This estimation is expected to speed up the optimization if the fitness landscape is still similar during the subsequential optimization tasks, is anticipated to be true if the goal parameters would not changed too much.

The focus of the experiments was to find out how well the optimizer can handle the changes in the goal parameters, how much fewer samples would be needed for a solution when using the sequential sampler and if the solution is still as optimal as without the sequential sampler. The expectation was also that at least with small sample counts the sequential sampler would perform better, allowing the animator to quickly get a decent preview of the transition animation which is being generated. Besides this, we studied how increasing the priority of a goal would affect the convergence times and the fitness of the resulting trajectory.

10.1 Test Setup

The tests tried several ways of modifying the goal parameters for generating a transition animation between a standing and a crouching animation. In all test cases, unless it is specified otherwise, the target root position of the character was first set to be one footstep away in the front of the starting position.

In the first 2 test cases, after evaluating 4000 samples with the optimizer, one of the following modifications to the goals was made and the optimization was signaled that the fitness landscape had changed:

- Test case 1: The target root position of the character at the end pose was changed to be 2 or 3 footsteps away from the starting position.
- Test case 2: The target root rotation of the character at the end pose was turned 22.5° or 45° counter-clockwise.

In the subsequent test cases, the priority of a goal was gradually increased by changing a σ_g parameter after each run of 4000 samples:

- Test case 3: The priority of the jerk minimization was gradually increased by changing the σ_J parameter of the jerk minimization goal from 200 to 20.
- Test case 4: The priority of matching the target joint positions at the end pose was increased by gradually changing the σ_P parameter of the pose goal from 0.5 to 0.1.
- Test case 5: The priority of matching the target root position at the end pose was increased by gradually changing the σ_R parameter of the pose goal from 0.5 to 0.1.
- Test case 6: The priority of the animation matching the target velocity at the end pose was increased by gradually changing σ_V parameter of the pose goal from 0.5 to 0.1.

The values were selected from the range of values which had generated the most visually pleasing results during the preliminary testing, but ranges of values which resulted in trivial optimization cases were tried to be avoided. Each of the test cases were also run a second time without using the sequential sampling method, so that the goal parameters were also modified straight away instead of first evaluating 4000 samples. During the test cases 3, 4 and 6 the goal for matching the desired root location at the end pose was removed.

The highest fitness value found after each sample evaluation was recorded. To overcome the randomness of the results of a stochastic optimizer, the test cases 1 and 2 were run 50 times and the rest of the test cases were run 15 times. After that the results of the runs were averaged among each test case. During a run, finding a better trajectory with a particular sample also caused sudden jumps in the graphs, which would still be present after the averaging. Therefore the curves were also smoothed to remove the jumps using the Formula 10.1, where $f_s(s)$ is the smoothed fitness curve, s is the sample index, a is the smooth factor ($a = 300$) and $f_b(s_o)$ returns the fitness of the best trajectory found after the evaluation of the sample s_o.

$$f_s(s) = \sum_{d=-a}^{a} \left(\left(1 - \frac{|d|}{a} \right) f_b(s + d) \right) \tag{10.1}$$

The smoothing was done because the jumps in the graph would be misleading as they would not be present with a sufficiently large number of runs, due to then being smoothed out by the averaging. Overall a larger number or runs would have naturally given even better results, as some jumps in the fitness value are still apparent even with the smoothing, but the long computing time required for each test case and the limited amount of time to run the tests limited the number of runs. Finally the results were presented with a logarithm scale for the fitness value, in order to show the relative increase in the fitness value even with the smallest values. The vertical axes of the graphs were cut to show the relevant information. It should be noted that the apparent gradual increase in the fitness value during the evaluation of the completely random samples ($s < 250$) was due to the smoothing. Random samples very rarely produced any decent results, but were necessary in the initialization of the optimization.

10.2 Target Root Position

In the first test case, the target position of the root of the character at the end pose was changed to be further away. This simulates the case when the animator wants to adjust how far the character should end up from the starting position during the motion. In the first test case, the target end position of the root of the character was first changed to be 2 footsteps away from the starting position and in the second case changed to be 3 footsteps away. The sequential sampler was first initialized by running 4000 samples without changing the target end position from being one footstep away from the start. Then the sequential sampler was signaled that the fitness landscape

was changed. After that the test cases were run in the same way both with the sequential and the non-sequential sampler. Then another run was made with the sequential sampler by first initializing it by running 4000 samples, during which the starting position changed to be 2 footsteps away from the starting position. Otherwise this second run was similar to the first run with the sequential sampler.

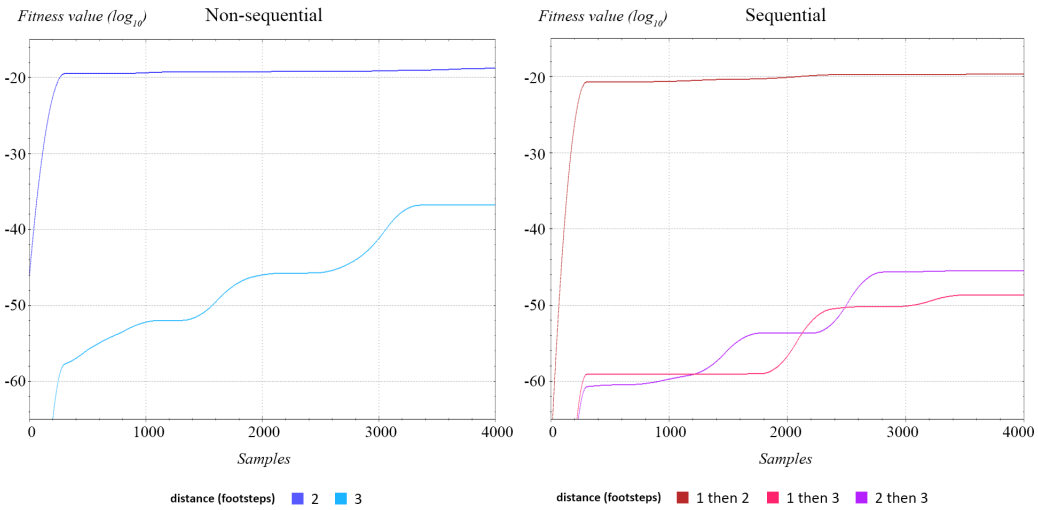

Figure 10.1: Fitness value development in the test case 1 with the sequential and the non-sequential sampler. Values are a smoothed average of 50 runs and the first 250 samples were drawn randomly. In the case of the sequential sampler, the sampler was first initialized by running 4000 samples with a different target distance, then the optimization was run with the actually tested target distance (e.g. first 2, then 3 footsteps away).

The results were quite similar in the 2 footsteps away case, the non-sequential sampler finding slightly better results overall (Figure 10.1). Both samplers were able to find a solution almost instantly and they could not find any significantly better trajectory later on, which also shows that the sequential sampler does not give any benefit when the non-sequential sampler is also able to find a decent solution quickly.

The difference was much bigger in the 3 footsteps away case, which was expected to be more difficult for the optimizer. The sequential sampler performed generally far worse, which could be because the trajectory to end up 3 footsteps away would be completely different than when the character only needs to take one or two steps. Therefore assuming that the fitness landscape

is still similar has likely misguided the sampler. This was especially noticeable when the sequential sampler was initialized by first running it with the target end position being first one footstep away (curve *1 then 3* in Figure 10.1), where it took the sequential sampler around 2000 samples to find any better results after the first 250 random samples. This could have been because the sequential optimizer needs to spend time re-evaluating some of the samples from the previous run, which did not help it to find suitable samples in this case. Initializing the sequential sampler with the target position being 2 footseps away (curve *2 then 3* in Figure 10.1) seems to have slightly helped, which is likely because the initialization target was closer to the actual target of ending up 3 footsteps away. However, the improvement was not large, even though in that case the sequential optimizer was not stuck in the first found solution for a very long time.

10.3 Target Root Rotation

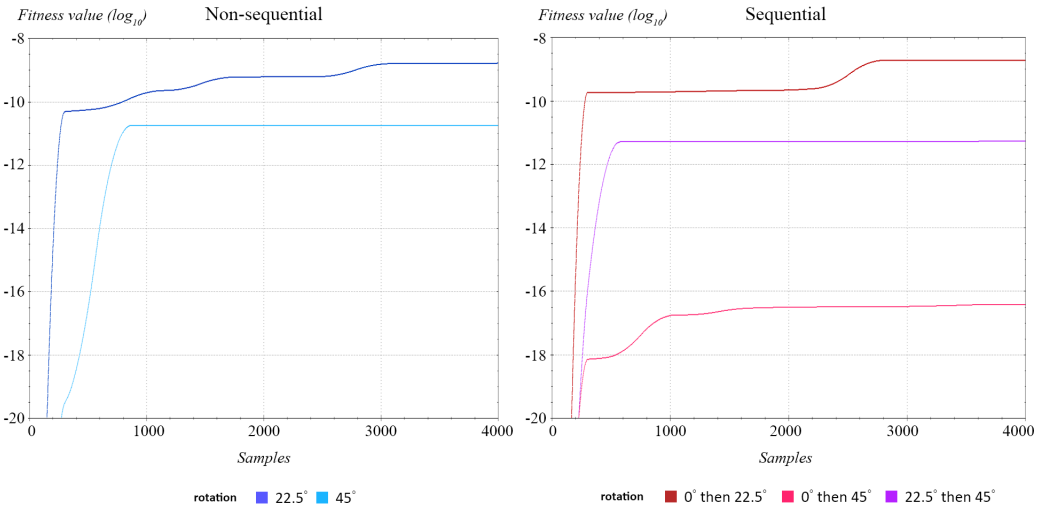

Figure 10.2: Fitness value development in the test case 2. Values are a smoothed average of 50 runs and the first 250 samples were drawn randomly.

In the second test case, the target rotation of the root of the character at the end pose was changed to be 22.5° or 45° counter-clockwise compared to the starting pose. Therefore the heading of the character should change 22.5° or 45° during the transition animation. A similar case to this test might arise when the animator would want to generate a crouching animation which

matches the direction of the obstacle that the character is attempting to hide behind. Several transition animations with different target directions could be generated in order to have a matching animation regardless of the starting rotation of the character in relation with the obstacle. There could therefore be a benefit in using the sequential sampler, which could attempt to use the fitness landscape of the previously generated animation to quickly generate a new one with slightly different heading for the end pose. When running the test cases with the sequential sampler, it was first initialized by running 4000 samples using the same target root rotation as the starting rotation, after which the sampler was signaled that the fitness landscape was changed. After that the actual test case was run normally.

The sequential sampler found a better solution immediately in both cases, with 22.5° and 45° turns (Figure 10.2). However, in the 22.5° case, both samplers ended up with similar results, and overall the performance was quite identical. The main difference was that the sequential sampler was stuck with finding only very small improvements to the trajectory until after evaluating around 2500 samples. This could be because the assumed shape of the fitness landscape would not encourage it to explore other maxima. However, because a way to plot the fitness landscape was not implemented, this could not be investigated.

In contrast to that, largely different results were had in the 45° case, when the sequential sampler was initialized by running 4000 samples without requiring any turning. When initialized in this way, with this kind of a more difficult optimization problem, the sequential sampler performed far worse than the non-sequential one. This was similar to the 3 footsteps away case in the previous section, and the reason for the worse results is likely also the same, which is that the 45° rotation is so large that it requires a completely different trajectory than when the heading of the character should not change at all. In this test case, however, initializing the sequential sampler first with 4000 samples with the target heading set to 22.5°, which is midway between 0° and 45°, helped the sequential optimizer significantly. In this case the sequential sampler almost matched the results of the non-sequential sampler. The reason for this could be that not turning at all is a completely different kind of a trajectory than when taking at least some kind of a turn in the same direction.

10.4 Jerk Minimization Priority

The third test case tested the effects of gradually increasing the priority of the jerk minimization. When a sequential sampler was used, the optimization

would first be run with σ_J set to 200. Then the optimizer would sequentially run the optimization task with σ_J first changed to 155, then 110, then 65 and finally 20. When the sequential sampler was not used, the optimization for each σ_J value was run separately.

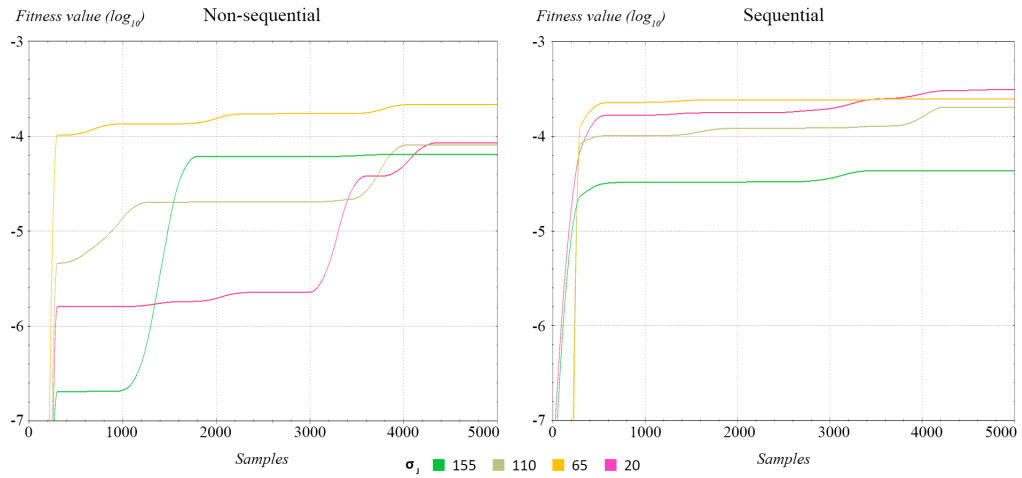

Figure 10.3: Fitness value development in the test case 3. Values are a smoothed average of 15 runs and the first 250 samples were drawn randomly.

As shown by Figure 10.3, the sequential sampler performed overall significantly better even with higher sample counts. It was able to find solutions with much higher fitness values right after the first 250 random samples than if it simply used the best sample from the previous run. Only in the first run ($\sigma_J = 155$) the non-sequential sampler was able to find a slightly better solution and was also able to produce rather close results with $\sigma_J = 65$. The better start for each run was expected since the sequential sampler knows the best trajectory with the previous σ_J value, which should have a decent fitness even with σ_J lowered. However, the starting fitness was significantly higher than predicted by the best sample from the previous run.

The fitness value after evaluating 4000 samples was sometimes higher and sometimes lower with a lower σ_J (Figure 10.3). This was surprising since a found trajectory should always have a higher fitness with a higher σ_J. The sequential sampler has obviously the advantage that it remembers the results from the previous run which may improve its convergence, but the results were similar in the $\sigma_J = 65$ case with the non-sequential sampler as well. It is possible that the higher weight in the jerk minimization has helped to lead the optimizer to a better solution.

The sequential sampler did not find largely better solutions after the first 500 samples, which might suggest that the optimizer would have got stuck in the initial solution it has found. However, it could also be that since the initially found solution is relatively good, a better solution is hard to find. In each case, though, an even better solution is eventually found, but much later after 3000–4000 sample evaluations.

10.5 Joint Position Priority

Sometimes the character could end up too far from the desired end pose due to other goals than the end pose matching having a relatively high priority. To guide the optimizer to find trajectories that better match the joint locations of the end pose, σ_P of the pose goal could be lowered. The test case 4 tested the effects of gradually lowering σ_P from 0.5 to 0.1 both with and without a sequential sampler. The tests were run similarly as in the third test case.

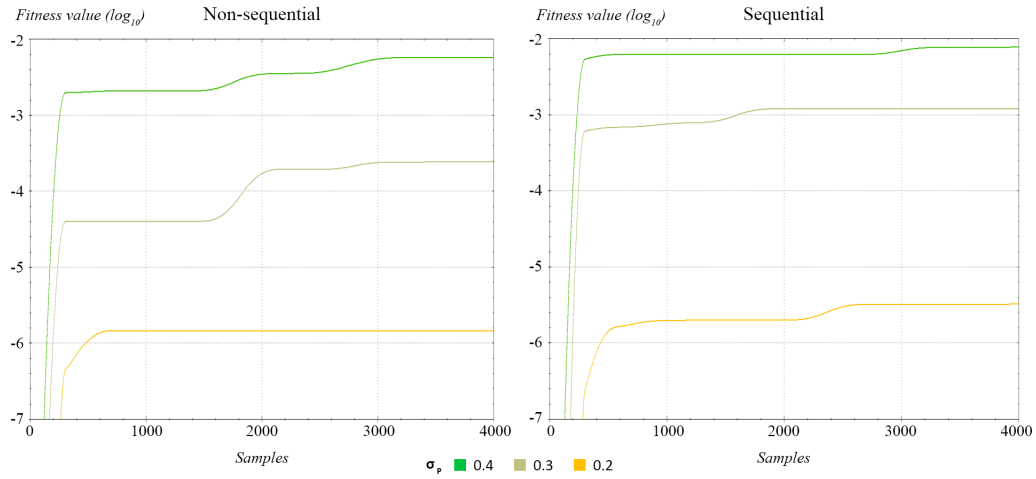

Figure 10.4: Fitness value development in the test case 4, with σ_P set to 0.4, 0.3 and 0.2. Values are a smoothed average of 15 runs and the first 250 samples were drawn randomly.

In every case the sequential sampler initially found samples with a higher fitness, as was expected (Figure 10.4 and Figure 10.5). The final results after 4000 samples were not largely different with the sequential and the non-sequential sampler except in the last case ($\sigma_P = 0.1$), where the sequential sampler was able to find a significantly better sample initially and where the

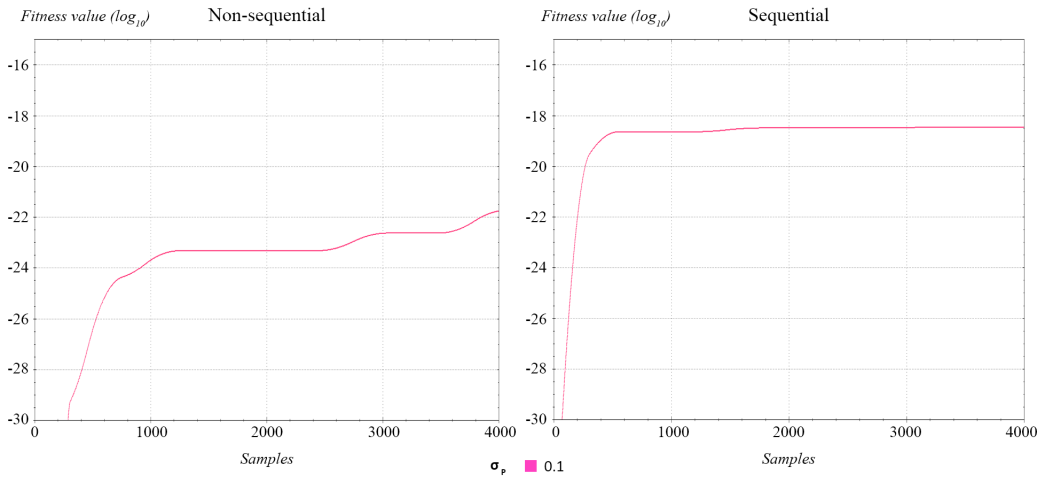

Figure 10.5: Fitness value development in the test case 4, with σ_P set to 0.1. Values are a smoothed average of 15 runs and the first 250 samples were drawn randomly.

non-sequential sampler was not able to find as good of a solution even with the whole sample budget. This could be because the sequential sampler has been able to gradually improve the trajectory over all the 5 subsequential runs instead of having to starting over.

10.6 Root Position Priority

In this example case of generating crouching animations, the character might end up too close or too far away from the obstacle behind which it should be hiding. The animator could attempt to fix this by increasing the priority of matching the correct root position when the transition animation is finished, by lowering the σ_R of the pose goal. In the 5th test case we gradually lowered σ_R from 0.5 to 0.1 similarly to the previous test case.

The sequential sampler largely outperformed the non-sequential version in this test case (Figure 10.6). Both the initial and the final fitness values were higher in every case. As in the third test case, the sequential sampler does not find a largely better solution after the first 500 samples before having evaluated over 3000 samples. However, in this case, the non-sequential sampler was also stuck in the same solution after 1000 samples and could also not find a better trajectory before evaluating around 3000 samples, which was similar when using the sequential sampler. It also found results with a sur-

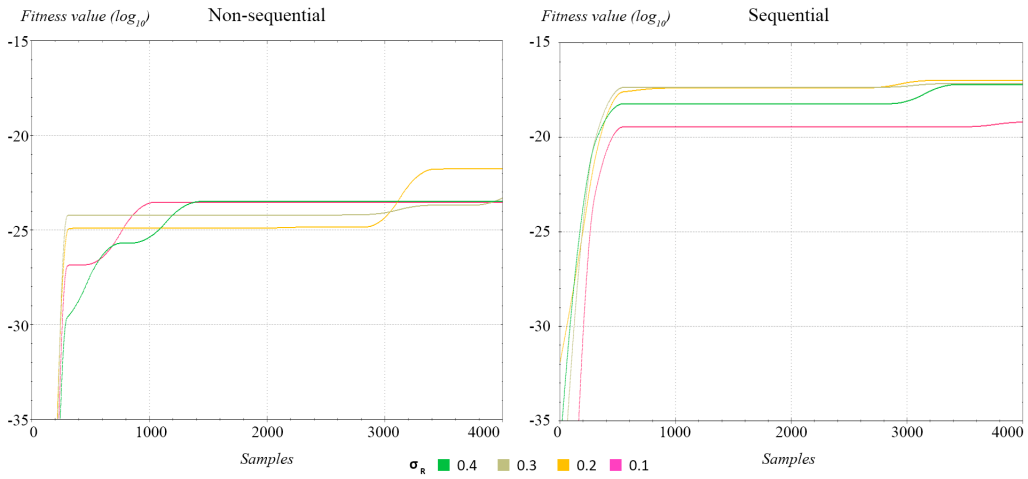

Figure 10.6: Fitness value development in the test case 5. Values are a smoothed average of 15 runs and the first 250 samples were drawn randomly.

prisingly similar fitness value regardless of the value of σ_R. The expectation was that the fitness value would be lower with a lower σ_R.

10.7 Velocity Matching Priority

If the velocity of the character at the end pose does not match the starting velocity of the character in the target animation, the motion could appear discontinuous. The σ_V parameter of the velocity matching goal can be lowered to increase the priority of matching the correct velocity when the transition animation ends. The 6th test case gradually lowered the value of the σ_V parameter of the pose goal from 0.5 to 0.1 similarly as in the previous test cases.

In both cases the optimizer was able to reach the solution right after the random samples were evaluated and the kd-tree based sampling started (Figure 10.7). The sequential sampler was however able to converge right away in every other case except the last one (σ_V=0.1), and even in that case it was able to find even better results right away than the non-sequential sampler could find with the entire sample budget. In that case the final fitness value was also significantly lower when using the non-sequential sampler.

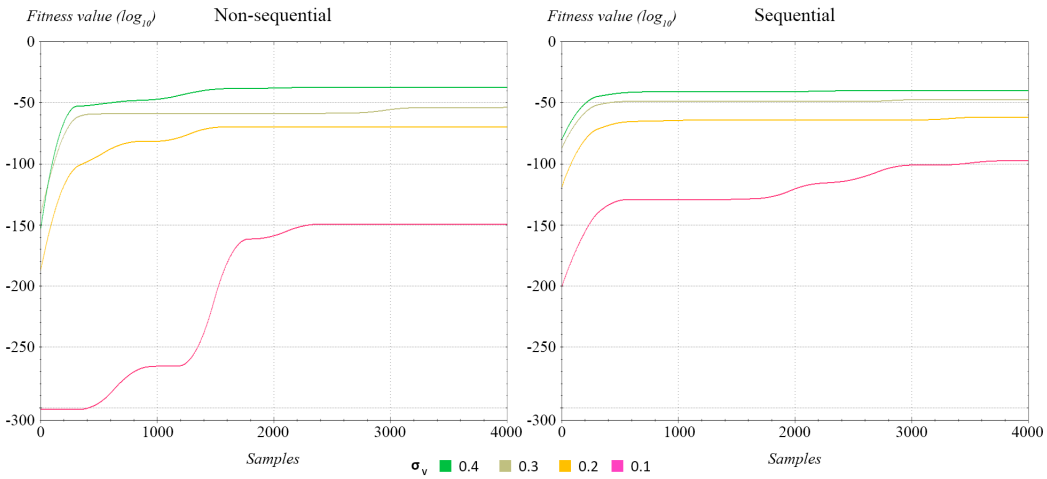

Figure 10.7: Fitness value development in the test case 6 with and without using a sequential sampler. The results are an average of 15 runs and the first 250 samples were drawn randomly.

10.8 Evaluation

Sequential sampling did not give any substantial benefit when the target position or rotation of the end pose were changed. This could be because those changes required the optimizer to find substantially different trajectories. The harder it was for the optimizer to find a decent solution, the less the sequential sampler helped. Even in the cases where the sequential sampler performed best, the non-sequential sampler found even better results. Without an option to visualize or otherwise analyze the high-dimensional fitness landscapes, the cause for this cannot be further analyzed.

The sequential sampler out-performed the non-sequential sampler in almost every case when the priority of a goal was gradually increased. As expected, it could find a decent solution almost instantly and that solution was also generally better than the trajectory that the non-sequential sampler was able to find during the entire run of 4000 samples. The almost immediately found solution was also considerably better than the best sample from the previous run. The harder finding a decent solution was, the better the sequential sampler performed in comparison with the non-sequential sampler in the test cases 3 to 6. However, there were also signs that the sequential sampler might have got stuck in the initial solution that it found and could not improve it until much later, though when starting with a relatively high-

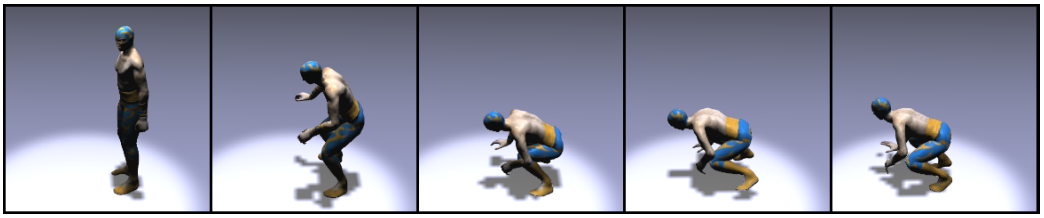

Figure 10.8: A generated transition animation between a standing and a crouching animation, without any desired movement or rotation for the root of the character. Even though the results may look similar to linear blending, the movement of the hands, feet and the head of the character takes in account that the character needs to remain balanced at all times.

fitness solution, finding a better one is likely also harder. It should be noted that the sequential sampler does not naturally offer any benefit for the first run, but only after the parameters of the goals have been changed.

In most test cases, the higher the priority of a goal was set to, the harder it was to find a sample with as high a fitness value. This was expected, since if the priority of a goal would be lowered, an already found trajectory should have a higher fitness value after the change. However, in some cases the optimizer was able to find a trajectory with a higher fitness value if the priority of a goal was increased. This happened so consistently that it is likely not just caused by lucky selection of samples by the stochastic optimizer. Instead, the increased priority might change the landscape in a way which has helped the optimizer to find a better solution, e.g. by reducing the amount of local minima in the landscape.

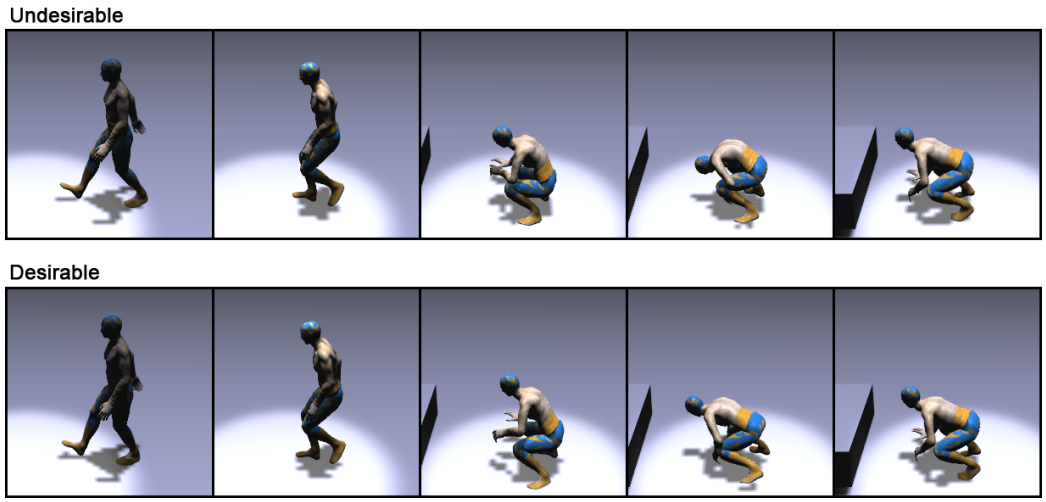

Figure 10.9: A generated transition animation between a walking and a crouching animation. The target position of the character in the end pose is defined to be close to the obstacle which the character should hide behind. Because the implementation does not define a sufficient set of stylistic goals, the results may sometimes be undesirable with the stochastic optimizer (above). In this case character has bent its neck way too much. However, increasing the priority of the jerk minimization led the optimization towards a more desirable result (below).

Chapter 11

Conclusions

A system was implemented for generating physically correct transition animations between arbitrary source and destination animations. The animator was still required to hand-tune some optimization parameters, but in case of the changing the goal priorities previewing the effect of the changes was made faster by using a sequential version of the importance sampler. Generally the stochastic optimization was able to find decent motion trajectories significantly faster than other similar works, such as by Al Borno et al. [2013], though the results are not directly comparable. The stochastic optimizer did not also get stuck in the first promising solution, but still often took long to find an another maximum. Also in the cases when the sequential sampler performed worse, it was stuck in a far worse solution for a long time.

The style of the resulting motion was not analyzed, but a quick review indicated that the resulting motion was not highly natural (Figures 10.8 and 10.9). It was, however, considerably better than the results of a simple linear interpolation, as the source and destination animations in every tested case were not intended to be blended together. The lack of naturalness might be due to the limited physical model of the character, e.g. due to using motors instead of muscles and because the character was modelled with rigid bodies instead of soft bodies. Additional goals which would encourage natural looking motion might also be needed.

Chapter 12

Future Work

Even if the sequential optimization allows for quickly reviewing how changing the goal parameters affects the results, the animator may not know how to select the correct values. The system could generate several trajectories with slightly different goal parameter values and let the animator to pick the most suitable one. A display could also be implemented which would tell which of the goals is the most limiting factor in the currently displayed trajectory and which is preventing the optimization from finding a better solution. Implementing a way to reduce the dimensionality of the fitness landscape and to display it could also help finding what kinds of issues the optimizer is facing with the landscape. Building a desired environment around the character is also difficult. Right now the editor allows only for creating boxes by using the scene editor in Unity, and only when the application is not running. Automatically importing the unity scene to the physical simulation could save a lot of work for the user.

Collisions between body parts were also not implemented, which required limiting the angles of the joints. This will likely result in movement which is not natural looking. Also the style of the movement was not largely taken in account. The only ways to change the style of the movement were defining maximum forces for the muscles, changing the priority of the jerk minimization and prioritizing movement which avoids hitting the head or the chest to an object. The system could attempt to derive the styling properties from the source and destination animations and interpolate them as well during the transition animation, possibly in a way similar to those introduced in Chapter 6.2. The naturalness of the resulting motion should also be qualitatively evaluated and the system should be improved according to the shortcomings that might be found.

Bibliography

M. Al Borno, M. de Lasa, and A. Hertzmann. Trajectory optimization for full-body movements with complex contacts. *Visualization and Computer Graphics, IEEE Transactions on*, 19(8):1405–1414, Aug 2013. ISSN 1077-2626. doi: 10.1109/TVCG.2012.325.

Ilya Baran and Jovan Popovic. Automatic rigging and animation of 3d characters. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. doi: 10.1145/1275808.1276467. URL `http://doi.acm.org/10.1145/1275808.1276467`.

Adrian Boeing and Thomas Bräunl. Evaluation of real-time physics simulation systems. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 281–288, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-912-8. doi: 10.1145/1321261.1321312. URL `http://doi.acm.org/10.1145/1321261.1321312`.

O. Brock, J.C. Trinkle, and F. Ramos. *Robotics: Science and Systems IV*. MIT Press, 2009. ISBN 9780262513098. URL `http://www.google.fi/books?id=fvCaQfBQ7qEC`.

Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. *SIGGRAPH Comput. Graph.*, 22(4):309–315, June 1988. ISSN 0097-8930. doi: 10.1145/378456.378531. URL `http://doi.acm.org/10.1145/378456.378531`.

David R. Carrier, Christoph Anders, and Nadja Schilling. The musculoskeletal system of humans is not tuned to maximize the economy of locomotion. *Proceedings of the National Academy of Sciences*, 108(46):18631–18636, 2011. doi: 10.1073/pnas.1105277108. URL `http://www.pnas.org/content/108/46/18631.abstract`.

Marco Da Silva, Yeuhi Abe, and J Popović. Simulation of human motion data using short-horizon model-predictive control. In *Computer Graphics Forum*, volume 27, pages 371–380. Wiley Online Library, 2008.

Marco da Silva, Yeuhi Abe, and Jovan Popovic, J. Interactive simulation of stylized human locomotion. *ACM Trans. Graph.*, 27(3):82:1–82:10, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360681. URL `http://doi.acm.org/10.1145/1360612.1360681`.

Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 251–260, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383287. URL `http://doi.acm.org/10.1145/383259.383287`.

Roy Featherstone. *Rigid body dynamics algorithms*, volume 49. 2008.

N. Pronost A. Egges Geijtenbeek, T. and M. H. Overmars. Interactive character animation using simulated physics. *Eurographics - State of the Art Reports*, 2, 2011.

Perttu Hämäläinen, Timo Aila, Tapio Takala, and Jarmo Alander. Mutated kd-tree importance sampling. SCAI, 2006.

Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 153–162, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7. doi: 10.1145/258734.258822. URL `http://dx.doi.org/10.1145/258734.258822`.

J.K. Hodgins and W.L. Wooten. Animating human athletes. In Yoshiaki Shirai and Shigeo Hirose, editors, *Robotics Research*, pages 356–367. Springer London, 1998. ISBN 978-1-4471-1582-3. doi: 10.1007/978-1-4471-1580-9_34. URL `http://dx.doi.org/10.1007/978-1-4471-1580-9_34`.

John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–72, 1992.

Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints. *SIGGRAPH Comput. Graph.*, 21(4):215–224, August 1987. ISSN 0097-8930. doi: 10.1145/37402.37428. URL `http://doi.acm.org/10.1145/37402.37428`.

Sumit Jain, Yuting Ye, and C. Karen Liu. Optimization-based interactive motion synthesis. *ACM Trans. Graph.*, 28(1):10:1–10:12, February 2009. ISSN 0730-0301. doi: 10.1145/1477926.1477936. URL `http://doi.acm.org/10.1145/1477926.1477936`.

Ladislav Kavan and Jiří Žára. Spherical blend skinning: A real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 9–16, New York, NY, USA, 2005. ACM. ISBN 1-59593-013-2. doi: 10.1145/1053427.1053429. URL `http://doi.acm.org/10.1145/1053427.1053429`.

M.M. Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, 1999. doi: doi:10.1016/S0959-4388(99)00028-8. URL `http://www.ingentaconnect.com/content/els/09594388/1999/00000009/00000006/art00028`.

James Kennedy, Russell Eberhart, et al. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 214–224, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5. URL `http://dl.acm.org/citation.cfm?id=846276.846307`.

Frances Y. Kuo and Ian H. Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52:1320–1329, 2005.

PeterJ.M. Laarhoven and EmileH.L. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, volume 37 of *Mathematics and Its Applications*, pages 7–15. Springer Netherlands, 1987. ISBN 978-90-481-8438-5. doi: 10.1007/978-94-015-7744-1_2. URL `http://dx.doi.org/10.1007/978-94-015-7744-1_2`.

C. Karen Liu and Zoran Popovic, J. Synthesis of complex dynamic character motion from simple animations. *ACM Trans. Graph.*, 21(3):408–416, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566596. URL `http://doi.acm.org/10.1145/566654.566596`.

C. Karen Liu, Aaron Hertzmann, and Zoran Popovic, J. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans.*

Graph., 24(3):1071–1081, jul 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073314. URL http://doi.acm.org/10.1145/1073204.1073314.

Janzen Lo, Gang Huang, and Dimitris Metaxas. Human motion planning based on recursive dynamics and optimal control techniques. *Multibody System Dynamics*, 8(4):433–458, 2002. ISSN 1384-5640. doi: 10.1023/A:1021111421247. URL http://dx.doi.org/10.1023/A%3A1021111421247.

J. McLester and P.S. Pierre. *Applied Biomechanics: Concepts and Connections*. Cengage Learning, 2007. ISBN 9780495105862. URL http://books.google.fi/books?id=JsFvOy1w-1QC.

S. Menardais, F. Multon, R. Kulpa, and B. Arnaldi. Motion blending for real-time animation while accounting for the environment. In *Computer Graphics International, 2004. Proceedings*, pages 156–159, June 2004. doi: 10.1109/CGI.2004.1309206.

L. Milic and Y. McConville. *The Animation Producer's Handbook*. McGraw-Hill Education, 2006. ISBN 9780335220366. URL http://www.google.fi/books?id=1W74AAAAQBAJ.

Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90 – 126, 2006. ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2006.08.002. URL http://www.sciencedirect.com/science/article/pii/S1077314206001263. Special Issue on Modeling People: Vision-based understanding of a person's shape, appearance, movement and behaviour.

Ádám Moravánszky. Novodex demo exercise 2005, 2005. URL http://bulletphysics.com/ftp/pub/test/physics/papers/NovodexSDK4.pdf.

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 71:1–71:8, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0210-4. doi: 10.1145/1833349.1778808. URL http://doi.acm.org/10.1145/1833349.1778808.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.*, 31 (4):43:1–43:8, July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185539. URL http://doi.acm.org/10.1145/2185520.2185539.

Masahiro Mori. The uncanny valley. *Energy*, 7(4):33–35, 1970.

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.*, 28 (3):81:1–81:9, July 2009. ISSN 0730-0301. doi: 10.1145/1531326.1531387. URL `http://doi.acm.org/10.1145/1531326.1531387`.

Michiel Panne and Alexis Lamouret. Guided optimization for balanced locomotion. In Demetri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, Eurographics, pages 165–177. Springer Vienna, 1995. ISBN 978-3-211-82738-3. doi: 10.1007/978-3-7091-9435-5_13. URL `http://dx.doi.org/10.1007/978-3-7091-9435-5_13`.

Rick Parent. *Computer Animation*. Morgan Kaufmann Publishers Inc., 2002.

Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5. doi: 10.1145/311535.311536. URL `http://dx.doi.org/10.1145/311535.311536`.

Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 501–508, New York, NY, USA, 2002. ACM. ISBN 1-58113-521-1. doi: 10.1145/566570.566608. URL `http://doi.acm.org/10.1145/566570.566608`.

C. Rose, M.F. Cohen, and B. Bodenheimer. Verbs and adverbs: multidimensional motion interpolation. *Computer Graphics and Applications, IEEE*, 18(5):32–40, Sep 1998. ISSN 0272-1716. doi: 10.1109/38.708559.

Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 147–154, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: 10.1145/237170.237229. URL `http://doi.acm.org/10.1145/237170.237229`.

Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 514–521, New York, NY, USA, 2004. ACM. doi: 10.1145/1186562.1015754. URL `http://doi.acm.org/10.1145/1186562.1015754`.

L. Sentis and O. Khatib. A whole-body control framework for humanoids operating in human environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2641–2648, May 2006. doi: 10.1109/ROBOT.2006.1642100.

A. James Stewart and James F. Cremer. Beyond keyframing: An algorithmic approach to animation. In *Proceedings of the Conference on Graphics Interface '92*, pages 273–281, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 0-9695338-1-0. URL `http://dl.acm.org/citation.cfm?id=155294.155326`.

Robert F. Stärk. Formal specification and verification of the c# thread model. *Theoretical Computer Science*, 343(3):482 – 508, 2005. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2005.06.028. URL `http://www.sciencedirect.com/science/article/pii/S0304397505003695`. Formal Methods for Components and Objects Formal Methods for Components and Objects 2003.

Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 291–300, New York, NY, USA, 2005. ACM. ISBN 1-59593-198-8. doi: 10.1145/1073368.1073410. URL `http://doi.acm.org/10.1145/1073368.1073410`.

Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913, Oct 2012. doi: 10.1109/IROS.2012.6386025.

H. Van Welbergen, B. J. H. Van Basten, A. Egges, Zs. M. Ruttkay, and M. H. Overmars. Real time animation of virtual humans: A trade-off between naturalness and control. *Computer Graphics Forum*, 29(8):2530–2554, 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2010.01822.x. URL `http://dx.doi.org/10.1111/j.1467-8659.2010.01822.x`.

Robert Y. Wang, Kari Pulli, and Jovan Popović. Real-time enveloping with rotational regression. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. doi: 10.1145/1275808.1276468. URL `http://doi.acm.org/10.1145/1275808.1276468`.

Andrew Witkin and Michael Kass. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pages 159–168, New York, NY, USA,

1988. ACM. ISBN 0-89791-275-6. doi: 10.1145/54852.378507. URL http://doi.acm.org/10.1145/54852.378507.

Jia-chi Wu and Zoran Popovic. Terrain-adaptive bipedal locomotion control. *ACM Trans. Graph.*, 29(4):72:1–72:10, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778809. URL http://doi.acm.org/10.1145/1778765.1778809.

KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Continuation methods for adapting simulated skills. *ACM Trans. Graph.*, 27(3):81:1–81:7, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360680. URL http://doi.acm.org/10.1145/1360612.1360680.

Victor Zordan, Adriano Macchietto, Jose Medin, Marc Soriano, Chun-Chih Wu, Ronald Metoyer, and Robert Rose. Anticipation from example. In *Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology*, VRST '07, pages 81–84, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-863-3. doi: 10.1145/1315184.1315197. URL http://doi.acm.org/10.1145/1315184.1315197.